

## Lab 2. Switch Circuit Control

### 1. Purpose

Lab 2의 목적은 GPIO hardware를 이용하여 다양한 방법으로 solenoid magnet을 control 하는데 있다. sysfs를 이용한 command line, shell script, 그리고 C++ program을 이용하여 이를 수행한다. LED를 사용한 기본적인 실험들을 통해 해당 과정이 어떻게 이루어지는 익하고 solenoid magnet을 이용한 실험을 수행한다. Solenoid magnet의 경우 추후 robotiv manipulator arm의 end effector(i.,e. gripper)로 이용된다.

- 안전한 GPIO control을 위한 간단한 switching circuit을 구성한다.
- Software(linux kernel)을 이용해 어떻게 hardware에 access하는지 이해한다.
- GPIO를 다음을 이용해 control 해본다.
  - 1) command using sysfs
  - 2) shell script
  - 3) C++ file executable

### 2. Experiment Sequence

본 lab은 네가지 problem으로 구성되어 있다. 전체적인 lab의 내용은 Beaglebone GPIO와 transistor array를 이용하여 switching circuit을 구현하고 이를 light controller와 solenoid magnet controller로 활용하는 것이다. Switching circuit은 electric switch operation을 수행하는 transistor가 포함된 wired circuit을 의미한다. LED light controller 구현을 위해서는 두개의 LED를 switching circuit에 연결하고 sys file system(sysfs), shell script, C++ program을 이용해 동작 시킨다. Solenoid magnet controller 구현을 위해서는 solenoid magnet을 switching circuit에 연결하고 C++ program을 이용해 동작 시킨다. 아래는 구현하게 될 네가지 problem이다.

#### (1) Problem 2A

GPIO를 이용해 두개의 LED를 연결하고, transistor array와 resistor도 함께 연결한다. Commands with sysfs를 이용해 Beaglebone 내장 LED인 USR0 LED를 control 해본다. 그 다음 commands with sysfs를 이용해 연결한 두개의 LED를 control 해본다.

#### (2) Problem 2B

Shell script를 이용해 switching circuit에 연결한 두 개의 LED를 control 해본다.

#### (3) Problem 2C

C++ program을 이용해 switching circuit에 연결한 두 개의 LED를 control 해본다.

#### (4) Problem 2D

Switching circuit에 solenoid magent과 transistor array, resistors를 연결한다. C++ program을 이용해

solenoid magnet을 control 해본다.

### 3. Experiment Results

#### Step 1. User LED0 control using sysfs and command line

가장 먼저 PC와 Beaglebone을 키고 연결하였으며 NFS와 ssh를 이용해 cross-development environment를 세팅하였다. sysfs에 대해 이해하기 위해 Beaglebone Black에 내장되어 있는 user LED인 LED0를 control 해본다. LED0의 기본 세팅은 heartbeat signal로 심장박동과 유사하게 켜지는 것을 확인할 수 있었다. User LED의 sysfs 파일을 확인하였다. Leds라는 directory에서 4개의 user LED 동작을 관할하는 directory를 확인할 수 있었다. 각 directory는 sysfs와 GPIO에 의한 system information을 제공한다.

```

alvinjinsung@beaglebone: ~
File Edit View Search Terminal Help
alvinjinsung@beaglebone:~/nfs_client$ ls
사진 음악 문서 공개 비디오 템플릿 바탕화면 다운로드 DesignLab snap
alvinjinsung@beaglebone:~/nfs_client$ cd ~
alvinjinsung@beaglebone:~$ sudo su
root@beaglebone:/home/alvinjinsung# ls -F /sys/class
ata_device/ extcon/ mdio_bus/ rfkill/ tty/
ata_link/ firmware/ mem/ rtc/ typec/
ata_port/ gnss/ misc/ scsi_device/ ubi/
backlight/ gpio/ mmc_host/ scsi_disk/ udc/
bdi/ graphics/ mtd/ scsi_generic/ uio/
block/ hidraw/ net/ scsi_host/ usb_role/
bsg/ hwmon/ phy/ sound/ vc/
devcoredump/ i2c-adapter/ power_supply/ spidev/ vtconsole/
devfreq/ i2c-dev/ pps/ spi_master/ watchdog/
devfreq-event/ input/ ptp/ spi_slave/
dma/ iommu/ pwm/ thermal/
drm/ leds/ regulator/ tpm/
drm_dp_aux_dev/ mbox/ remoteproc/ tpmrm/
root@beaglebone:/home/alvinjinsung# ls -F /sys/class/leds
beaglebone:green:usr0@ beaglebone:green:usr2@
beaglebone:green:usr1@ beaglebone:green:usr3@
root@beaglebone:/home/alvinjinsung# 

```

그림 1. Beaglebone Black의 sysfs 파일과 user LED 동작을 관할하는 directory

LED0에 해당하는 directory에 접근해 LED0의 동작을 조절해보았다. ls -F 명령어를 통해 brightness, invert, power/, trigger, device@, max\_brightness, subsystem@, uevent등의 옵션들이 있는 것을 확인하였다. trigger에 대한 정보를 확인하기 위해 cat trigger 명령어를 사용하였고, 현재 value가 heartbeat으로 지정되어 있는 것을 확인하였다. None 값을 trigger에 지정해주며 LED 동작을 멈추었다.

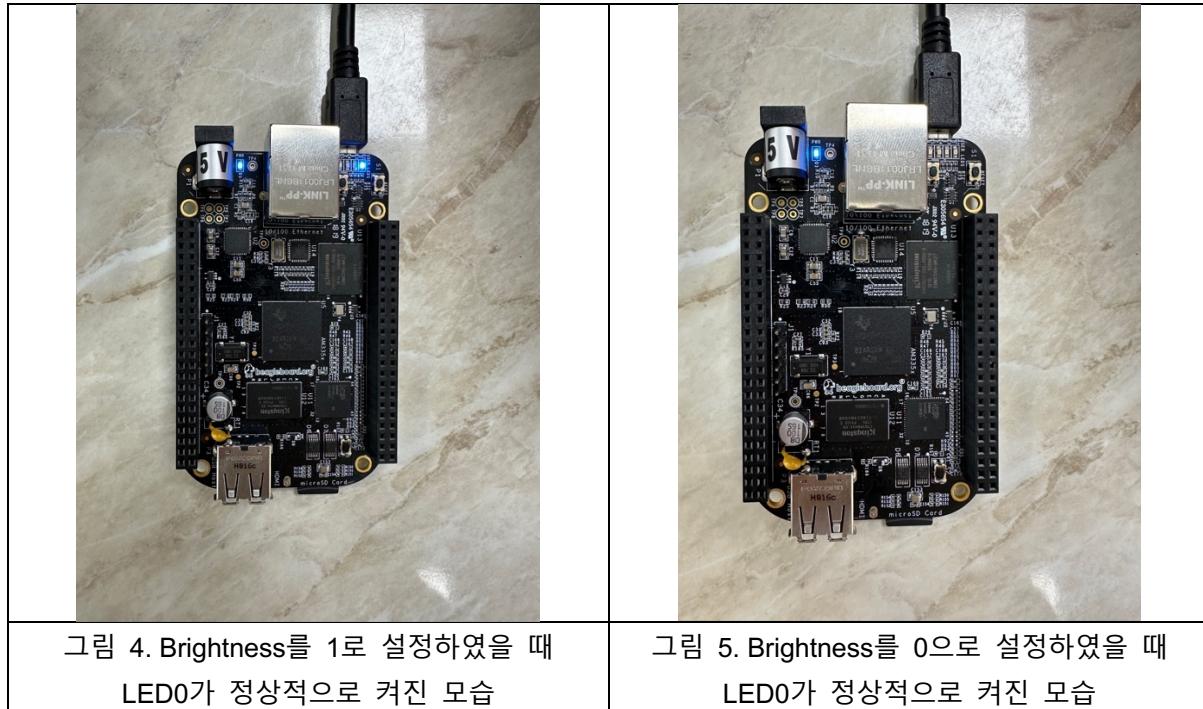
```
alvinjinsung@beaglebone: ~
File Edit View Search Terminal Help
root@beaglebone:/home/alvinjinsung# cd /sys/class/leds
root@beaglebone:/sys/class/leds# cd beaglebone\:green\:usr0
root@beaglebone:/sys/class/leds/beaglebone:green:usr0# ls -F
brightness invert power/ trigger
device@ max_brightness subsystem@ uevent
root@beaglebone:/sys/class/leds/beaglebone:green:usr0# cat trigger
none rfkill-any rfkill-none kbd-scrolllock kbd-numlock kbd-capslock kbd-kanalock
kbd-shiftlock kbd-altgrlock kbd-ctrllock kbd-altlock kbd-shiftlock kbd-shiftr
ock kbd-ctrllock 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
root@beaglebone:/sys/class/leds/beaglebone:green:usr0# echo none > trigger
root@beaglebone:/sys/class/leds/beaglebone:green:usr0# 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-shiftlock kbd-shift
rlock kbd-ctrllock 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
root@beaglebone:/sys/class/leds/beaglebone:green:usr0#
```

그림 2. User LED0 directory에 접근하여 trigger 정보를 확인하고 변형해본 결과

다음은 brightness 값을 1로 설정하여 LED를 키고, 0으로 설정하여 LED를 끄는 작업을 진행하였다. User LED0가 정상적으로 켜졌다 꺼지는 것을 확인할 수 있었다.

```
alvinjinsung@beaglebone: ~
File Edit View Search Terminal Help
root@beaglebone:/sys/class/leds/beaglebone:green:usr0# echo 1 > brightness
root@beaglebone:/sys/class/leds/beaglebone:green:usr0# echo 0 > brightness
root@beaglebone:/sys/class/leds/beaglebone:green:usr0#
```

그림 3. Brightness 값을 1과 0으로 설정한 모습



다음은 trigger를 timer로 설정하고 delay\_on을 100, delay\_off를 900으로 설정하여 LED가 켜지는 사이에 delay를 주는 명령을 실행해 보았다. LED가 정상적으로 깜빡이며 켜지면서 dealy가 생기는 것을 확인할 수 있었다. 이 후 다시 default 값인 heartbeat signal로 trigger를 설정하였다.

```
alvinjinsung@beaglebone: ~
File Edit View Search Terminal Help
root@beaglebone:/sys/class/leds/beaglebone:green:usr0# echo timer > trigger
root@beaglebone:/sys/class/leds/beaglebone:green:usr0# echo 100 > delay_on
root@beaglebone:/sys/class/leds/beaglebone:green:usr0# echo 900 > delay_off
root@beaglebone:/sys/class/leds/beaglebone:green:usr0#
```

그림 6. LED trigger를 timer로 설정하고 delay를 설정한 모습

```
alvinjinsung@beaglebone: ~
File Edit View Search Terminal Help
root@beaglebone:/sys/class/leds/beaglebone:green:usr0# echo heartbeat > trigger
root@beaglebone:/sys/class/leds/beaglebone:green:usr0#
```

그림 7. LED trigger를 default값인 heartbeat로 설정한 모습

### Step 2(Problem 2A). Switching circuit control using sysfs and command line

본 실험을 진행하기 위해서 switching circuit을 Beaglebone Black에 연결하였다. 두개의 GPIO를 이용해 두개의 LED를 control 하며 GPIO\_30과 GPIO\_31을 이용하였다. 회로 연결은 아래와 같이 이루어졌다.

- Wiring for Light 1:  
GPIO0\_30(P9.11) → R1 → uLN2803AN B input; uLN2803AN OC output → R2 → Cathode of LED; Anode of LED → 5V
- Wiring for Light 2:
- GPIO0\_31(P9.13) → R1 → uLN2803AN B input; uLN2803AN OC output → R2 → Cathode of LED; Anode of LED → 5V

R1의 경우  $5k\Omega$ 으로 설정하였으며, GPIO의 output이 0V 또는 5V이기에 1mA 이하의 안전한 전류를 보장한다. R2의 경우 연결 전압이 5V이며 LED로 인한 전압강하가 3.5V라고 가정할 때, 30mA의 전류 세기를 보장하기 위해  $(5V-3.5V)/R2 = 30mA$ 를 만족해야 하며, R2는 50Ω이면 본 식을 만족한다. 51Ω 저항을 이용해 회로를 구성하였다. 완성된 회로의 모습은 아래와 같다.

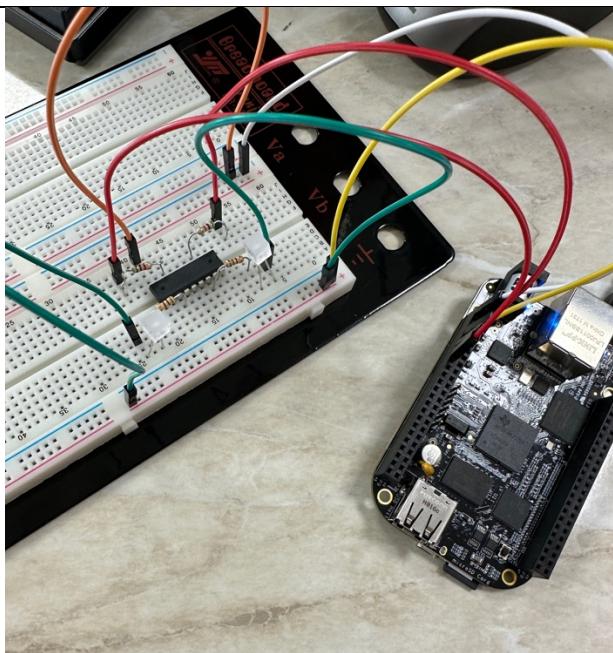
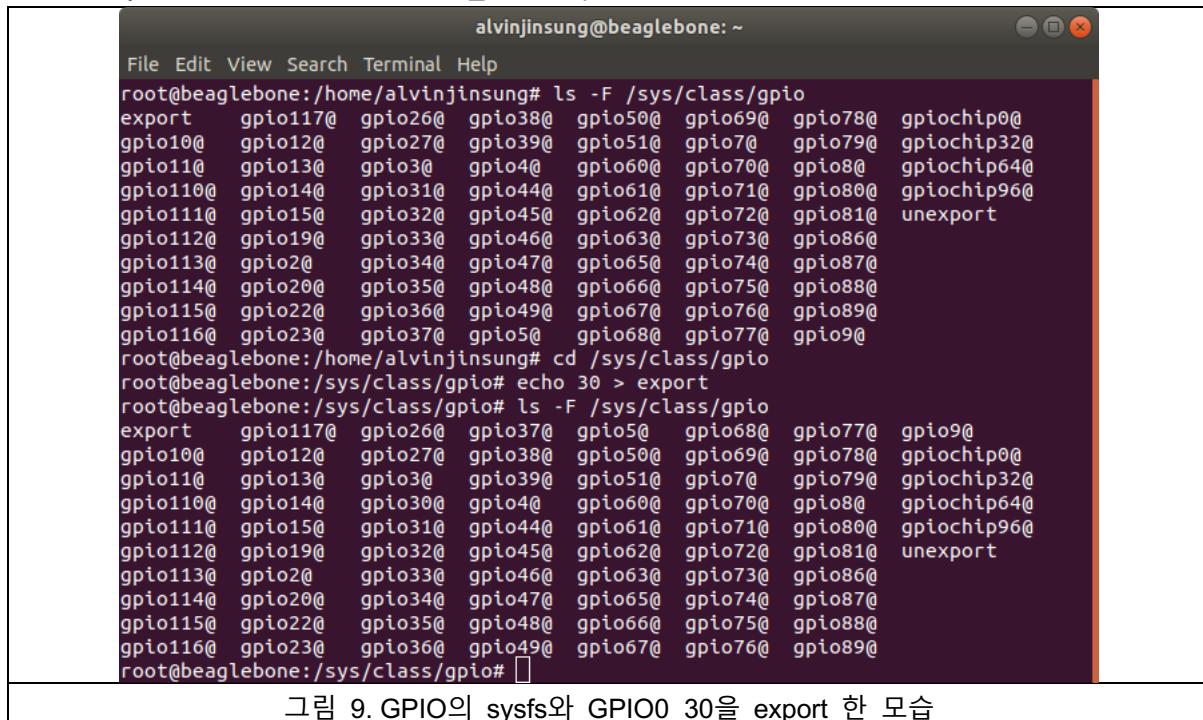


그림 8. Problem 2B에 해당하는 switching circuit을 GPIO에 연결한 모습

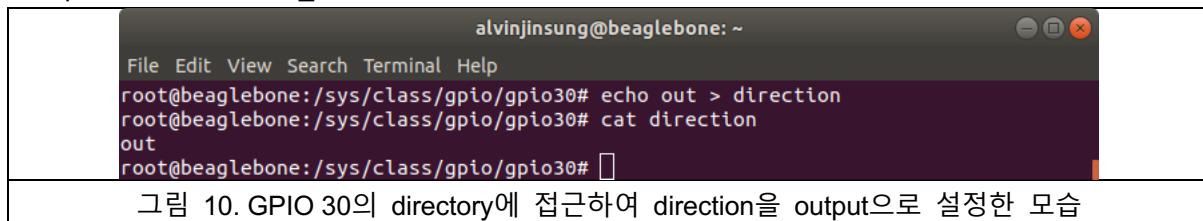
GPIO의 sysfs를 확인하였으며 GPIO0\_30을 export하였다.



```
alvinjinsung@beaglebone: ~
File Edit View Search Terminal Help
root@beaglebone:/home/alvinjinsung# ls -F /sys/class/gpio
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@  gpio31@  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@ 
root@beaglebone:/home/alvinjinsung# cd /sys/class/gpio
root@beaglebone:/sys/class/gpio# echo 30 > export
root@beaglebone:/sys/class/gpio# ls -F /sys/class/gpio
export  gpio117@  gpio26@  gpio37@  gpio5@   gpio68@  gpio77@  gpio9@ 
gpio10@  gpio12@  gpio27@  gpio38@  gpio50@  gpio69@  gpio78@  gpiochip0@
gpio11@  gpio13@  gpio3@   gpio39@  gpio51@  gpio7@   gpio79@  gpiochip32@
gpio110@  gpio14@  gpio30@  gpio4@   gpio60@  gpio70@  gpio8@   gpiochip64@
gpio111@  gpio15@  gpio31@  gpio44@  gpio61@  gpio71@  gpio80@  gpiochip96@
gpio112@  gpio19@  gpio32@  gpio45@  gpio62@  gpio72@  gpio81@  unexport
gpio113@  gpio2@   gpio33@  gpio46@  gpio63@  gpio73@  gpio86@ 
gpio114@  gpio20@  gpio34@  gpio47@  gpio65@  gpio74@  gpio87@ 
gpio115@  gpio22@  gpio35@  gpio48@  gpio66@  gpio75@  gpio88@ 
gpio116@  gpio23@  gpio36@  gpio49@  gpio67@  gpio76@  gpio89@ 
root@beaglebone:/sys/class/gpio#
```

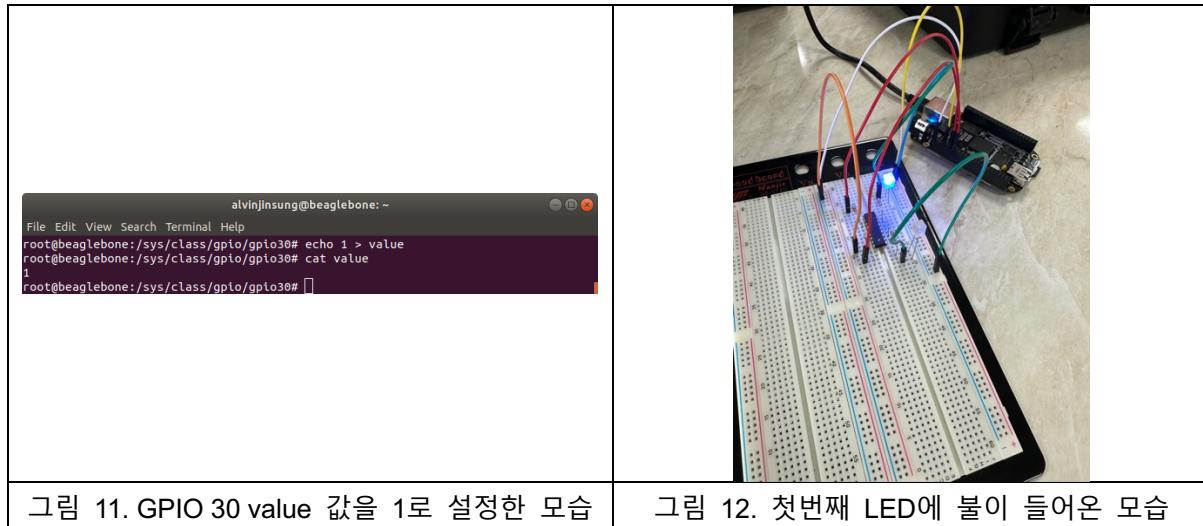
그림 9. GPIO의 sysfs와 GPIO0\_30을 export 한 모습

이후 GPIO 30 directory에 접근하여 첫번째 LED를 control 해 보았다. GPIO의 direction을 output으로 지정하였으며, value 값을 1과 0으로 지정하여 LED를 키고 꺼보았다. 본 과정을 거친 후에는 unexport를 통해 GPIO0\_30을 free 하였다.



```
alvinjinsung@beaglebone: ~
File Edit View Search Terminal Help
root@beaglebone:/sys/class/gpio/gpio30# echo out > direction
root@beaglebone:/sys/class/gpio/gpio30# cat direction
out
root@beaglebone:/sys/class/gpio/gpio30#
```

그림 10. GPIO 30의 directory에 접근하여 direction을 output으로 설정한 모습



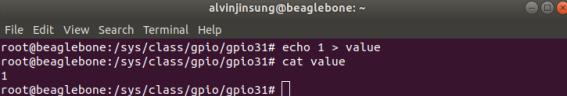
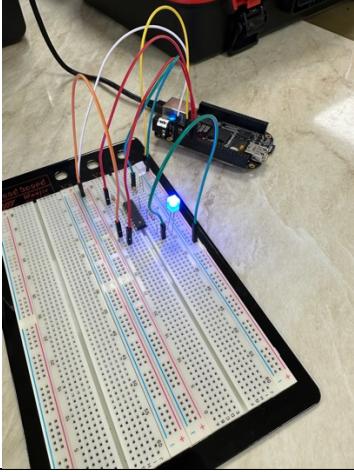
<pre>alvinjinsung@beaglebone: ~ File Edit View Search Terminal Help root@beaglebone:/sys/class/gpio/gpio30# echo 0 &gt; value root@beaglebone:/sys/class/gpio/gpio30# cat value 0 root@beaglebone:/sys/class/gpio/gpio30#</pre>	
그림 13. GPIO 30 value 값을 0로 설정한 모습	그림 14. 첫번째 LED에 불이 꺼진 모습

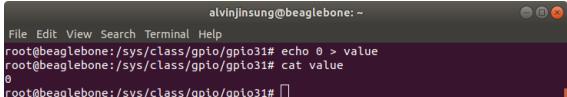
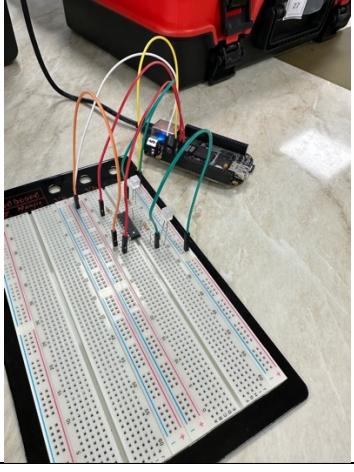
<pre>alvinjinsung@beaglebone: ~ File Edit View Search Terminal Help root@beaglebone:/sys/class/gpio/gpio30# cd /sys/class/gpio root@beaglebone:/sys/class/gpio# echo 30 &gt; unexport root@beaglebone:/sys/class/gpio#</pre>
그림 15. GPIO0_30을 unexport한 모습

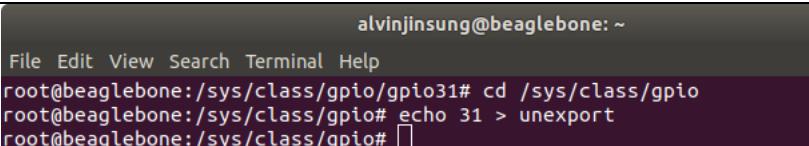
GPIO0\_31에 대해서도 위의 과정을 동일하게 진행해 주었다.

<pre>alvinjinsung@beaglebone: ~ File Edit View Search Terminal Help root@beaglebone:/sys/class/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@   gpio80@ gpio112@  gpio19@  gpio34@  gpio47@  gpio65@  gpio74@  gpio87@   gpio87@ gpio113@  gpio2@   gpio35@  gpio48@  gpio66@  gpio75@  gpio88@   gpio88@ gpio114@  gpio20@  gpio36@  gpio49@  gpio67@  gpio76@  gpio89@   gpio89@ gpio115@  gpio22@  gpio37@  gpio5@   gpio68@  gpio77@  gpio9@   gpio90@ gpio116@  gpio23@  gpio38@  gpio50@  gpio69@  gpio78@  gpio90@   gpio90@ root@beaglebone:/sys/class/gpio# echo 31 &gt; export root@beaglebone:/sys/class/gpio# ls -F /sys/class/gpio 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@  gpio31@  gpio44@  gpio61@  gpio71@  gpio80@  gpiochip96@ gpio111@  gpio15@  gpio32@  gpio45@  gpio62@  gpio72@  gpio81@  unexport gpio112@  gpio19@  gpio33@  gpio46@  gpio63@  gpio73@  gpio86@  gpio86@ gpio113@  gpio2@   gpio34@  gpio47@  gpio65@  gpio74@  gpio87@  gpio87@ gpio114@  gpio20@  gpio35@  gpio48@  gpio66@  gpio75@  gpio88@  gpio88@ gpio115@  gpio22@  gpio36@  gpio49@  gpio67@  gpio76@  gpio89@  gpio89@ gpio116@  gpio23@  gpio37@  gpio5@   gpio68@  gpio77@  gpio9@   gpio90@ root@beaglebone:/sys/class/gpio#</pre>
그림 16. GPIO의 sysfs와 GPIO0_31을 export 한 모습

<pre>alvinjinsung@beaglebone: ~ File Edit View Search Terminal Help root@beaglebone:/sys/class/gpio# cd gpio31 root@beaglebone:/sys/class/gpio/gpio31# echo out &gt; direction root@beaglebone:/sys/class/gpio/gpio31# cat direction out root@beaglebone:/sys/class/gpio/gpio31#</pre>
그림 17. GPIO 31의 directory에 접근하여 direction을 output으로 설정한 모습

 <pre>alvinjinsung@beaglebone: ~ File Edit View Search Terminal Help root@beaglebone:/sys/class/gpio/gpio31# echo 1 &gt; value root@beaglebone:/sys/class/gpio/gpio31# cat value 1 root@beaglebone:/sys/class/gpio/gpio31#</pre>	
그림 18. GPIO 31 value 값을 1로 설정한 모습	그림 19. 두번째 LED에 불이 들어온 모습

 <pre>alvinjinsung@beaglebone: ~ File Edit View Search Terminal Help root@beaglebone:/sys/class/gpio/gpio31# echo 0 &gt; value root@beaglebone:/sys/class/gpio/gpio31# cat value 0 root@beaglebone:/sys/class/gpio/gpio31#</pre>	
그림 20. GPIO 31 value 값을 0로 설정한 모습	그림 21. 두번째 LED에 불이 깨진 모습

 <pre>alvinjinsung@beaglebone: ~ File Edit View Search Terminal Help root@beaglebone:/sys/class/gpio/gpio31# cd /sys/class/gpio root@beaglebone:/sys/class/gpio# echo 31 &gt; unexport root@beaglebone:/sys/class/gpio#</pre>
그림 22. GPIO0_31을 unexport한 모습

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

Problem 2B에서는 LED를 연결한 switching circuit을 shell script를 이용해 control하였다. 두 개의 shell script를 작성하였으며 이를 이용해 GPIO에 연결된 LED가 우리가 원하는 방식으로 동작하도록 하였다. `Ui_control_lights.sh`는 GPIO 30과 GPIO 31에 각각 연결된 LED를 1과 2라는 변수로 지정하고 `user input`으로 1 또는 1가 들어오고 `on`이나 `off` 명령이 들어오면 해당 LED에 해당 기능이 이루어지도록 작성되었다. 예를 들어 `user input`으로 “1 on”이 입력되었을 경우 GPIO 30에 연결된 LED를 1로 지정하였기에 해당 LED에 불이 들어오고 “2 on”이 입력되었을 경우 GPIO 31에 연결된 LED가 불이 들어오게 된다. 0을 입력할 경우 while loop를 빠져나오게 되며 프로그램을 `exit`하게 된다. 해당 shell script 코드와 실행시킨 결과는 아래와 같다.

```

#!/bin/bash

# ui_control_lights.sh
# This code tests for the time takes for LED light control for many loops.

echo "ui_control_lights.sh"

echo "Export: Get access permission for GPIO30 & 31"
echo 30 > /sys/class/gpio/export
echo 31 > /sys/class/gpio/export

echo "Set directions of GPIO 30 & 31 as output"
echo out > /sys/class/gpio/gpio30/direction
echo out > /sys/class/gpio/gpio31/direction

while true
do

read -p "Type 'light_id' and 'on_off'" lid on_off

if [ $lid -lt 1 ]
then
echo "Input light_id is less than 1: Exit"
break
fi

if [ $lid -gt 2 ]
then
echo "Error Light_ID: 0:Exit, 1:Light_1, 2:Light_2"
continue
fi

if [ $on_off != "on" -a $on_off != "off" ]
then
echo "Error ON/OFF: Please type 'on' or 'off'"
continue
fi

if [ $lid -eq 1 ]
then
    if [ $on_off == "on" ]
    then
        echo 1 > /sys/class/gpio/gpio30/value
    fi
    if [ $on_off == "off" ]
    then
        echo 0 > /sys/class/gpio/gpio30/value
    fi
fi
fi

if [ $lid -eq 2 ]
then
    if [ $on_off == "on" ]
    then
        echo 1 > /sys/class/gpio/gpio31/value
    fi
    if [ $on_off == "off" ]
    then
        echo 0 > /sys/class/gpio/gpio31/value
    fi
fi
fi

done

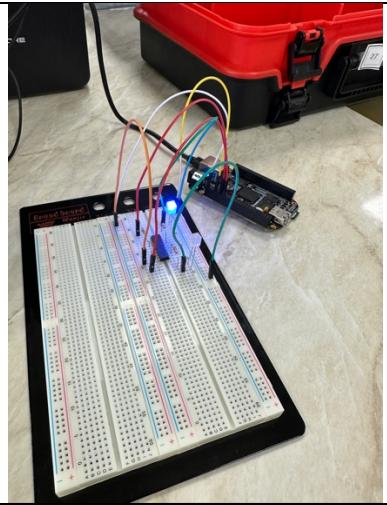
echo "Set directions of GPIO 30 & 31 as input"
echo in > /sys/class/gpio/gpio30/direction
echo in > /sys/class/gpio/gpio31/direction

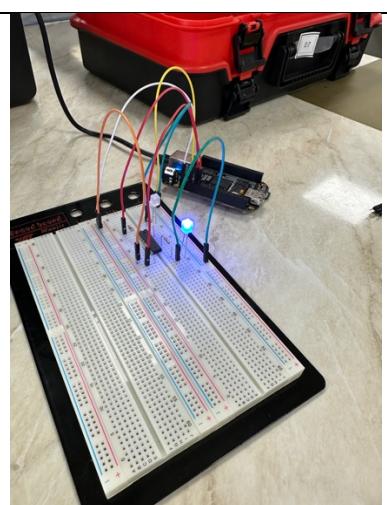
echo "UnExport: Release access permission for GPIO30 & 31"
echo 30 > /sys/class/gpio/unexport
echo 31 > /sys/class/gpio/unexport

```

그림 23. ui\_control\_lights.sh 코드

	<pre>kaist@kaist-n5: ~ File Edit View Search Terminal Help kaist@kaist-n5:~\$ mkdir -p ~/DesignLab/20170699/2_SwitchControl/b_LED_ShellScript kaist@kaist-n5:~\$ </pre>
그림 24. shell programming을 위한 directory를 생성한 모습	

	
그림 25. “1 on”을 user input으로 입력한 모습	

	
그림 26. GPIO 30에 연결된 LED가 켜진 모습	

다음은 `loop_control_lights.sh` 코드를 완성하고 실행하였다. 해당 프로그램은 두개의 LED가 켜졌다 꺼지는 것을 10번 반복하고 반복 시작 시간과 끝나는 시간을 기록하여 출력하는 프로그램이다. `ui_control_lights.sh`와 비슷하게 GPIO 30과 GPIO 31을 `export` 해주고 각각의 `direction`을 `output`으로 설정하였다. 그 다음 시간을 `start` 변수에 저장하고 `loop`에 들어가 두개의 LED를 켰다 끄는 과정을 10번 반복하였다. Loop를 빠져나와 시간을 `end` 변수에 저장하고 `start time`과 `end time`을 출력하였다. GPIO 30과 GPIO 31의 `direction`을 다시 `input`으로 지정하고 `unexport` 해준 다음 프로그램을 종료하였다. 해당 shell script 코드와 실행시킨 결과는 아래와 같다.

```

#!/bin/bash

# loop_control_lights.sh
# This code tests for the time takes for LED light control for many loops.

echo "loop_control_lights.sh"

# -----
# 1) set GPIO 30/31 as export
# !! REPLACE THIS PART TO YOUR CODE !!
echo "Export: Get access permission for GPIO30 & 31"
echo 30 > /sys/class/gpio/export
echo 31 > /sys/class/gpio/export

# -----
# 2) set GPIO 30/31 direction as output
# !! REPLACE THIS PART TO YOUR CODE !!
echo "Set directions of GPIO 30 & 31 as output"
echo out > /sys/class/gpio/gpio30/direction
echo out > /sys/class/gpio/gpio31/direction

# -----
# 3) save the start time(before the light control)
start=$(date +%s.%N)

for ((i=0;i<10;i++))
do

# -----
# 4) LED Light control loop - Please refer to the experiment guide
# !! REPLACE THIS PART TO YOUR CODE !!
    echo 1 > /sys/class/gpio/gpio30/value
    echo 1 > /sys/class/gpio/gpio31/value

    echo 0 > /sys/class/gpio/gpio30/value
    echo 0 > /sys/class/gpio/gpio31/value

# -----
done

# 5) save the end time(after the light control)
end=$(date +%s.%N)
echo "Start:$start*I End:$end"

# -----
# 6) set GPIO 30/31 direction as input
# !! REPLACE THIS PART TO YOUR CODE !!
echo "Set directions of GPIO 30 & 31 as input"
echo in > /sys/class/gpio/gpio30/direction
echo in > /sys/class/gpio/gpio31/direction

# -----
# 7) set GPIO 30/31 as unexport
# !! REPLACE THIS PART TO YOUR CODE !!
echo "UnExport: Release access permission for GPIO30 & 31"
echo 30 > /sys/class/gpio/unexport
echo 31 > /sys/class/gpio/unexport

```

그림 29. loop\_control\_lights.sh 코드

```
alvinjinsung@beaglebone: ~/nfs_client/DesignLab/20170699/2_SwitchControl/b_LED_ShellScript$ sudo chmod a+x loop_control_lights.sh
alvinjinsung@beaglebone: ~/nfs_client/DesignLab/20170699/2_SwitchControl/b_LED_ShellScript$ sudo ./loop_control_lights.sh
loop_control_lights.sh
Export: Get access permission for GPIO30 & 31
Set directions of GPIO 30 & 31 as output
Start:1677785408.527157005*I End:1677785408.565163963
Set directions of GPIO 30 & 31 as input
UnExport: Release access permission for GPIO30 & 31
alvinjinsung@beaglebone: ~/nfs_client/DesignLab/20170699/2_SwitchControl/b_LED_ShellScript$
```

그림 30. Loop\_control\_lights.sh 실행 결과

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

Problem 2C에서는 problem 2B에서 진행한 task를 C++ 코드를 통해 작동시켰다. 파일 구조는 include 폴더 내에 gpio\_control.hpp 파일과 src 폴더 내에 gpio\_control.cpp, test\_light\_control.cpp, loop\_light\_control.cpp 파일로 이루어져 있다. gpio\_control.cpp 파일에는 특정 gpio를 조작하는데 필요한 function들을 생성해 주었다. gpio\_export(), gpio\_unexport(), gpio\_set\_dir(), gpio\_fd\_open(), gpio\_fd\_close(), gpio\_fd\_set\_value()의 function들을 정의해 두었다. gpio\_export() function은 gpio port number를 input으로 받아 해당 port를 export 해주는 역할을 한다. gpio\_unexport() function은 gpio port number를 input으로 받아 해당 port를 unexport 해주는 역할을 한다. gpio\_set\_dir() function은 gpio port number와 direction을 input으로 받아 해당 port의 direction을 지정해준다. 1이 입력으로 들어올 경우 direction을 output 방향으로, 0일 입력으로 들어올 경우 direction을 input 방향으로 설정한다. gpio\_fd\_open() function은 gpio port number를 input으로 받아 해당 port의 file descriptor info를 return 해준다. gpio\_fd\_close() function은 gpio file descriptor를 input으로 받아 file descriptor info를 이용해 해당 gpio port를 close 해준다. gpio\_fd\_set\_value() function은 gpio file descriptor와 value를 input으로 받아 value가 1일 경우 gpio를 ON으로 변경하고 value가 0일 경우 gpio를 OFF로 변경한다. 해당 코드는 아래와 같다.

```
#include "../include/gpio_control.hpp"

#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <unistd.h>

int gpio_export(unsigned int gpio)
{
    /*
     - input : GPIO port number to export
     - function that exports GPIO port

     !! MODIFY FOLLOWING SAMPLE CODE PROPERLY !!

    */
    int fd;
    char buf[MAX_BUF];
    fd = open("/sys/class/gpio/export", O_WRONLY);
    sprintf(buf, "%d", gpio);
    write(fd, buf, strlen(buf));
    close(fd);
    return 0;
}
```

```
}

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

 !! MODIFY FOLLOWING SAMPLE CODE PROPERLY !!

*/
int fd;
char buf[MAX_BUF];
fd = open("/sys/class/gpio/unexport", O_WRONLY);
sprintf(buf, "%d", gpio);
write(fd, buf, strlen(buf));
close(fd);
return 0;
}

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

 !! MODIFY FOLLOWING SAMPLE CODE PROPERLY !!

*/
int fd;
char buf[MAX_BUF];
sprintf(buf, "/sys/class/gpio/gpio%d/direction", gpio);
fd = open(buf, O_WRONLY); //
if (direction == 1) {
    write(fd, "out", 3); // Set out direction
}
else if (direction == 0) {
    write(fd, "in", 2); // Set in direction
}
else {
    close(fd);
    return -1;
}
close(fd);
return 0;
}

int gpio_fd_open(unsigned int gpio)
{
/*
 - 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

 !! MODIFY FOLLOWING SAMPLE CODE PROPERLY !!

*/
int fd;
char buf[MAX_BUF];
sprintf(buf, "/sys/class/gpio/gpio%d/value", gpio);
fd = open(buf, O_RDWR | O_NONBLOCK);
return fd;
```

```

}

int gpio_fd_close(int fd)
{
/*
 - function which closes the GPIO port using file descriptor information
 - input : fd to close corresponding GPIO port
*/
    close(fd);
    return 0;
}

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

 !! MODIFY FOLLOWING SAMPLE CODE PROPERLY !!

*/
    if (value == 1) {
        write(fd, "1", 1); // Set GPIO ON
    }
    else if (value == 0) {
        write(fd, "0", 1); // Set GPIO OFF
    }
    else {
        return -1;
    }
    return 0;
}

```

그림 31. gpio\_control.cpp 코드

위의 function들을 사용하여 test\_light\_control.cpp 코드와 loop\_light\_control.cpp 코드를 완성하였다. test\_light\_control.cpp는 user input으로 light\_id와 “on” 또는 “off”를 입력받으면 해당 light\_id에 해당하는 gpio를 export하고 output direction으로 설정한 후 file descriptor info를 open하여 led를 키거나 끄는 동작을 수행한다. 프로그램을 종료할 때는 file descriptor info를 close하고 input으로 dirction을 설정한 후 gpio를 unexport한다. loop\_light\_control.cpp는 실행 시 두개의 led에 연결된 gpio를 export하고 output direction으로 설정한 후 file descriptor info를 open한다. 그 후 두개의 led 가 커졌다 꺼지는 것을 10번 반복하는데, 이때 실행 시작 시간과 끝나는 시간을 기록하여 출력한다. 그 다음 file descriptor info를 close하고 input으로 dirction을 설정한 후 gpio를 unexport하여 프로그램을 종료한다. 해당 코드는 아래와 같다.

```

#include <stdio.h>
#include <string.h>
#include "../include/gpio_control.hpp"

int main(int argc, char *argv[])
{
    int light_id;
    int fd_gpio_30, fd_gpio_31;
    int on_off_int;
    char on_off_str[8];

    /* 0.Print title */

```

```
printf("test_light_control\n");

/* 1. Set variables: light_id = 1; */
light_id = 1;

/*
2. Export GPIO 30 & GPIO31

!! REPLACE THIS PART TO YOUR CODE !!

*/

gpio_export(30);
gpio_export(31);

/*
3. Set direction of GPIO 30 & GPIO31 to out.

!! REPLACE THIS PART TO YOUR CODE !!

*/

gpio_set_dir(30, 1);
gpio_set_dir(31, 1);

/*
4. Open GPIO30 & GPIO31.

!! REPLACE THIS PART TO YOUR CODE !!

*/

fd_gpio_30 = gpio_fd_open(30);
fd_gpio_31 = gpio_fd_open(31);

/* 5. Loop while light_id > 0 */
while(light_id > 0)
{
    /* A. Prompt output */
    printf("Enter light_id and on_off_str: ");
    /* B. Get user input of light_id and on_off_str */
    scanf("%d %7s", &light_id, on_off_str);
    /* C. Check light id. Break if <= 0. */
    if(light_id <= 0)
        break;
    /* D. Check on_off_str and set on_off_int */
    if(!strcmp(on_off_str, "off"))
        on_off_int = 0;
    else if(!strcmp(on_off_str, "on"))
        on_off_int = 1;
    else
        on_off_int = -1;

    /*
5-E. Control the LED according to the user input

!! REPLACE THIS PART TO YOUR CODE !!

*/
    if (light_id == 30) {
        gpio_fd_set_value(fd_gpio_30, on_off_int);
    }

    else if (light_id == 31) {
        gpio_fd_set_value(fd_gpio_31, on_off_int);
    }
}

/*
6. Close GPIO30 & GPIO31

!! REPLACE THIS PART TO YOUR CODE !!

```

```

*/
    gpio_fd_close(fd_gpio_30);
    gpio_fd_close(fd_gpio_31);

/*
7. Set direction of GPIO30 & GPIO31 to in.

!! REPLACE THIS PART TO YOUR CODE !!

*/
    gpio_set_dir(30, 0);
    gpio_set_dir(31, 0);

/*
8. Unexport GPIO30 & GPIO31

!! REPLACE THIS PART TO YOUR CODE !!

*/
    gpio_unexport(30);
    gpio_unexport(31);

    return 0;
}

```

그림 32. test\_control\_light.cpp 코드

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/time.h>
#include "../include/gpio_control.hpp"

int main(int argc, char *argv[])
{
    int fd_gpio_30, fd_gpio_31;

    struct timeval s_time, e_time;
    struct timezone tz;
    double start_time, end_time;

    /* 0.Print title */
    printf("loop_light_control\n");

/*
1. Export GPIO 30 & GPIO31

!! REPLACE THIS PART TO YOUR CODE !!

*/
    gpio_export(30);
    gpio_export(31);

/*
2. Set direction of GPIO 30 & GPIO31 to out.

!! REPLACE THIS PART TO YOUR CODE !!

*/
    gpio_set_dir(30, 1);
    gpio_set_dir(31, 1);

/*
3. Open GPIO30 & GPIO31.

!! REPLACE THIS PART TO YOUR CODE !!

*/
    fd_gpio_30 = gpio_fd_open(30);
    fd_gpio_31 = gpio_fd_open(31);

```

```

// 4. Save the start time(before the light control loop)
gettimeofday(&s_time, &tz);

/*
5. LED Control loop for 10 times

!! REPLACE THIS PART TO YOUR CODE !!

*/

for (int i=0; i<10; i++) {
    gpio_fd_set_value(fd_gpio_30, 1);
    gpio_fd_set_value(fd_gpio_31, 1);

    gpio_fd_set_value(fd_gpio_30, 0);
    gpio_fd_set_value(fd_gpio_31, 0);
}

// 6. Save the end time(after the light control loop)

gettimeofday(&e_time, &tz);

start_time = s_time.tv_sec + s_time.tv_usec*1e-6;
end_time = e_time.tv_sec + e_time.tv_usec*1e-6;
printf("start: %.6f end: %.6f\n", start_time, end_time);

/*
7. Close GPIO30 & GPIO31

!! REPLACE THIS PART TO YOUR CODE !!

*/

gpio_fd_close(fd_gpio_30);
gpio_fd_close(fd_gpio_31);

/*
8. Set direction of GPIO30 & GPIO31 to in.

!! REPLACE THIS PART TO YOUR CODE !!

*/

gpio_set_dir(30, 0);
gpio_set_dir(31, 0);

/*
9. Unexport GPIO30 & GPIO31

!! REPLACE THIS PART TO YOUR CODE !!

*/

gpio_unexport(30);
gpio_unexport(31);

return 0;
}

```

그림 33. loop\_control\_light.cpp 코드

위 코드들을 이용해 testLight과 loopLight이란 두개의 executable을 build하였다. VSCode에서 CMakeLists.txt를 수정하여 cross-compile하였다. Compile한 결과는 아래와 같다.

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure under "C\_LED\_CPPPROGRAM".
- Code Editor:** Displays the CMakeLists.txt file content:

```
cmake_minimum_required(VERSION 3.0.0)
project(testLight VERSION 0.1.0)

include_directories(include)
add_executable(testLight src/test_light_control.cpp src/gpio_control.cpp)
```

- Terminal:** Shows the build command and its execution:

```
[main] Building Folder: c_LED_CppProgram
[build] Starting build
[proc] Executing command: /usr/bin/cmake --build /home/kaist/designLab/20170699_2_SwitchControl/c_LED_CppProgram/build --config Debug --target all -- -j 8
[build] [100%] Built target testLight
[build] Build finished with exit code 0
```

- Status Bar:** Shows "Ln 8, Col 1" and other build-related information.

그림 34. testLight cross-compile 결과

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure under "C\_LED\_CPPPROGRAM".
- Code Editor:** Displays the CMakeLists.txt file content:

```
cmake_minimum_required(VERSION 3.0.0)
project(loopLight VERSION 0.1.0)

include_directories(include)
add_executable(loopLight src/loop_light_control.cpp src/gpio_control.cpp)
```

- Terminal:** Shows the build command and its execution:

```
[main] Configuring project: c_LED_CppProgram
[cmake] Configuring done
[cmake] Generating done
[build] Starting build
[proc] Executing command: /usr/bin/cmake --build /home/kaist/designLab/20170699_2_SwitchControl/c_LED_CppProgram/build --config Debug --target all -- -j 8
[build] Scanning dependencies of target loopLight
[build] [ 66%] Building CXX object CMakeFiles/loopLight.dir/src/gpio_control.cpp.o
[build] [ 66%] Building CXX object CMakeFiles/loopLight.dir/src/loop_light_control.cpp.o
[build] [100%] Linking CXX executable loopLight
[build] [100%] Built target loopLight
[build] Build finished with exit code 0
```

- Status Bar:** Shows "Configuration is already in progress." and other build-related information.

그림 35. loopLight cross-compile 결과

Cross-compile이 성공적으로 완료되어 Beaglebone black에서 실행하며 led가 원하는대로 잘 동작하는지 test해보았다. 결과는 아래와 같다.

```
alvinjinsung@beaglebone: ~/nfs_client/DesignLab/20170699/2_SwitchControl/c_LED_CppPr... ● ● ○
File Edit View Search Terminal Help
root@beaglebone:/home/alvinjinsung/nfs_client/DesignLab/20170699/2_SwitchControl
/c_LED_CppProgram# ./build/testlight
test_light_control
Enter light_id and on_off_str: 30 on
Enter light_id and on_off_str: []
```

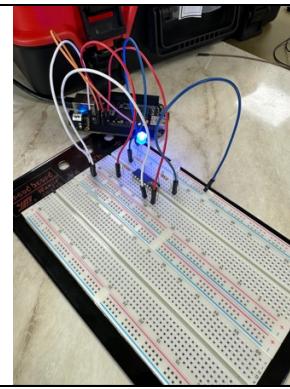


그림 36. “30 on”을 user input으로 입력한 모습  
(testLight)

그림 37. GPIO 30에 연결된 LED가 켜진 모습

```
alvinjinsung@beaglebone: ~/nfs_client/DesignLab/20170699/2_SwitchControl/c_LED_CppPr... ● ● ○
File Edit View Search Terminal Help
root@beaglebone:/home/alvinjinsung/nfs_client/DesignLab/20170699/2_SwitchControl
/c_LED_CppProgram# ./build/testlight
test_light_control
Enter light_id and on_off_str: 30 on
Enter light_id and on_off_str: 31 on
Enter light_id and on_off_str: []
```

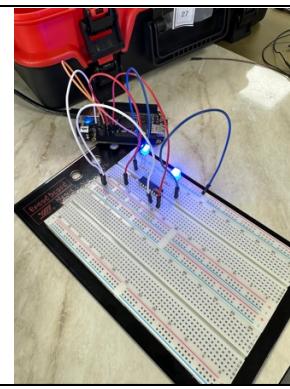


그림 38. “30 on, 31on”을 user input으로  
입력한 모습  
(testLight)

그림 39. GPIO 30, GPIO 31에 연결된  
LED가 켜진 모습

```
alvinjinsung@beaglebone: ~/nfs_client/DesignLab/20170699/2_SwitchControl/c_LED_CppPr... ● ● ○
File Edit View Search Terminal Help
root@beaglebone:/home/alvinjinsung/nfs_client/DesignLab/20170699/2_SwitchControl
/c_LED_CppProgram# ./build/testlight
test_light_control
Enter light_id and on_off_str: 30 on
Enter light_id and on_off_str: 31 on
Enter light_id and on_off_str: 30 off
Enter light_id and on_off_str: []
```

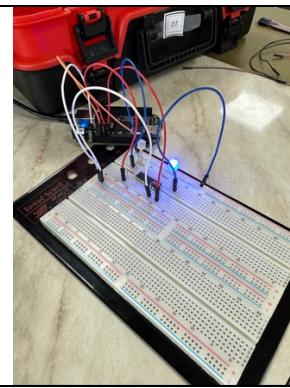
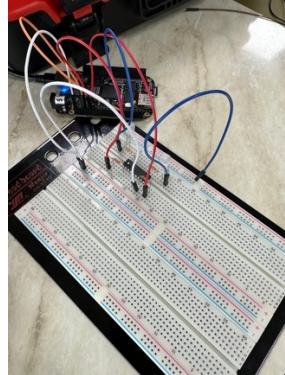


그림 40. “30 off, 31on”을 user input으로  
입력한 모습  
(testLight)

그림 41. GPIO 31에 연결된 LED가 켜진 모습

<pre>alvinjinsung@beaglebone: ~/nfs_client/DesignLab/20170699/2_SwitchControl/c_LED_CppPr... ◻ ◻ ◻ File Edit View Search Terminal Help root@beaglebone:/home/alvinjinsung/nfs_client/DesignLab/20170699/2_SwitchControl /c_LED_CppProgram# ./build/testLight test_light_control Enter light_id and on_off_str: 30 on Enter light_id and on_off_str: 31 on Enter light_id and on_off_str: 30 off Enter light_id and on_off_str: 31 off Enter light_id and on_off_str: ◻</pre>	
<p>그림 42. “30 off, 31off”을 user input으로 입력한 모습 (testLight)</p>	<p>그림 43. LED가 모두 꺼진 모습</p>

<pre>alvinjinsung@beaglebone: ~/nfs_client/DesignLab/20170699/2_SwitchControl/c_LED_CppPr... ◻ ◻ ◻ File Edit View Search Terminal Help root@beaglebone:/home/alvinjinsung/nfs_client/DesignLab/20170699/2_SwitchControl /c_LED_CppProgram# ./build/loopLight loop_light_control start: 1677792089.521483 end: 1677792089.521613 root@beaglebone:/home/alvinjinsung/nfs_client/DesignLab/20170699/2_SwitchControl /c_LED_CppProgram# ◻</pre>
<p>그림 44. loopLight실행 결과</p>

testLight은 user input대로 연결된 led가 켜지고 꺼졌으며 loopLight의 경우 10번 켜지고 꺼지는 동작을 진행하며 시작 시간과 종료 시간을 제대로 출력하였다.

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

Problem 2D에서는 C++ 코드를 통해 solenoid magnet을 우리가 원하는 대로 동작 시켰다. 이를 위해 우선 solenoid magnet을 포함한 circuit을 구성하였다. 회로 연결은 아래와 같이 이루어졌다.

- Electric switch input part:  
*GPIO0\_30(P9.11) → R3 → uLN2803AN B input*
- Power supply part:  
*12V Power supply(+) → R4 → Solenoid magnet\* → uLN2803AN C output*
- Common ground part:  
*GPIO GND → uLN2803AN GND; 12V Power supply(-) → uLN2803AN GND;*

R3의 경우  $5k\Omega$ 으로 설정하였으며, solenoid magnet의 appropriate voltage/current/power가 12V / 0.26A / 2.5W이기에 R4의 경우  $51\Omega$ 으로 설정하였다. 완성된 회로의 모습은 아래와 같다.

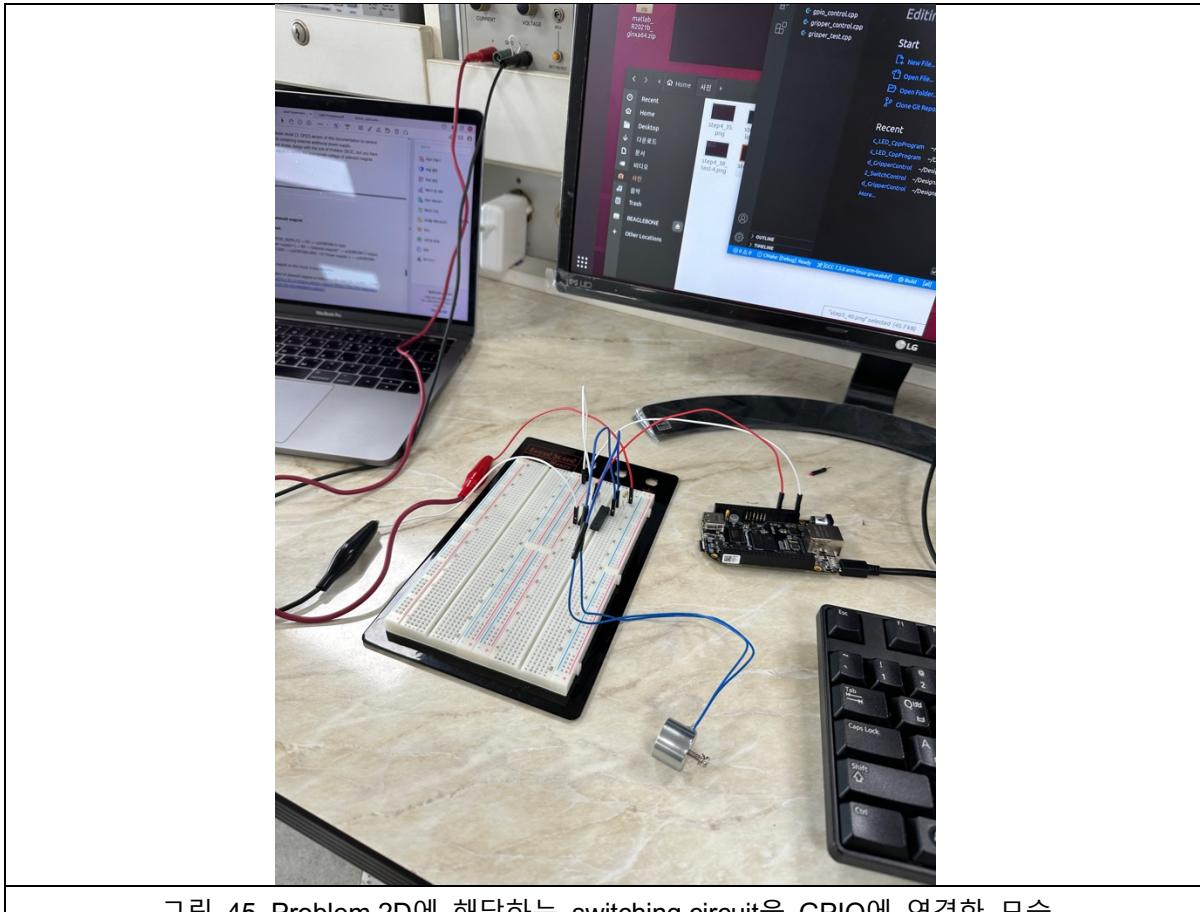


그림 45. Problem 2D에 해당하는 switching circuit을 GPIO에 연결한 모습

파일 구조는 include 폴더 내에 gpio\_control.hpp, gripper\_control.hpp 파일과 src 폴더 내에 gpio\_control.cpp, gripper\_control.cpp, gripper\_test.cpp 파일로 이루어져 있다. gpio\_control.hpp와 gpio\_control.cpp 파일은 problem 2C와 동일하며 gripper\_control.cpp 파일에는 solenoid를 control하기 위해 필요한 function들을 생성해 주었다. gripper\_open(), gripper\_on(), gripper\_off(), gripper\_close() function들을 정의해 두었다. gripper\_open() function은 gpio port number를 input으로 받아 해당 port의 file descriptor를 return한다. gripper\_on() function은 gpio port number를 input으로 받아 해당 port의 value를 1로 설정한다. gripper\_off() function은 gpio port number를 input으로 받아 해당 port의 value를 0으로 설정한다. gripper\_close() function은 gpio port number와 file descriptor를 input으로 받아 해당 port의 file descriptor를 close하고 gpio를 Input direction으로 바꾸고 unexport 한다. 해당 코드는 아래와 같다.

```
#include <stdio.h>
#include <string.h>
#include <iostream>
#include "../include/gpio_control.hpp"

int gripper_open(unsigned int gpio)
{
    /*
        open the GPIO port for the solenoid magnet
        - input : GPIO port number
        - output : file descriptor of GPIO port
    */
}
```

```

    !! FILL WITH YOUR CODE !!
 */

int fd_gpio;

gpio_export(gpio);
gpio_set_dir(gpio, 1);

fd_gpio = gpio_fd_open(gpio);

return fd_gpio;
}

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

   !! FILL WITH YOUR CODE !!
*/

gpio_fd_set_value(fd_gpio, 1);

return 0;
}

int gripper_off(unsigned int fd_gpio)
{
/*
   turn off the solenoid magnet
   - input : GPIO port number

   !! FILL WITH YOUR CODE !!
*/

gpio_fd_set_value(fd_gpio, 0);

return 0;
}

int gripper_close(unsigned int gpio, unsigned int fd_gpio)
{
/*

```

그림 46. gripper\_control.cpp 코드

gripper\_test.cpp은 gripper\_control.cpp의 function들을 사용하여 지정된 시간동안 solenoid magnet을 키고 지정된 시간이 지나면 solenoid magnet이 꺼지는 프로그램이다. 해당 코드는 아래와 같다.

```

#include <stdio.h>
#include <string.h>
#include <chrono>
#include "../include/gpio_control.hpp"
#include "../include/gripper_control.hpp"

int main(int argc, char *argv[])
{

/*
 1. Define variables for GPIO port number, GPIO file descriptor and length of time
duration for holding an object
  !! FILL WITH YOUR CODE !!
 */

int gpio = 30;
int fd_gpio;

printf("gripper_control\n");

/*
 2. Open the gripper GPIO port

```

```
!! FILL WITH YOUR CODE !!
*/
fd_gpio = gripper_open(gpio);

/* Save start time of the loop */
std::chrono::system_clock::time_point start_time = std::chrono::system_clock::now();
std::chrono::system_clock::time_point current_time;
std::chrono::microseconds loop_elapsed_time_microsec;

loop_elapsed_time_microsec =
std::chrono::duration_cast<std::chrono::microseconds>(current_time - start_time);
int pickup_time = 10;

/* Turn on the gripper while pickup_time */
while(loop_elapsed_time_microsec.count()/1e6 < pickup_time)
{
    current_time = std::chrono::system_clock::now();
    loop_elapsed_time_microsec =
std::chrono::duration_cast<std::chrono::microseconds>(current_time - start_time);
    /*
        3. Turn on the gripper
        !! FILL WITH YOUR CODE !!
    */

    gripper_on(fd_gpio);
}

/*
    4. Close the gripper GPIO port
    !! FILL WITH YOUR CODE !!
*/

gripper_off(fd_gpio);
gripper_close(gpio, fd_gpio);

return 0;
}
```

그림 47. gripper\_test.cpp 코드

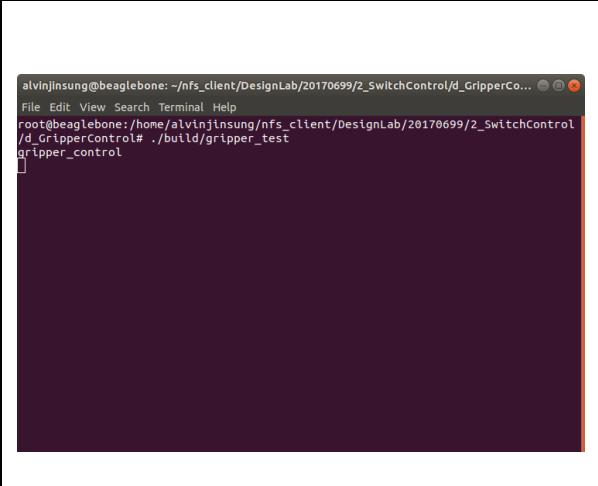
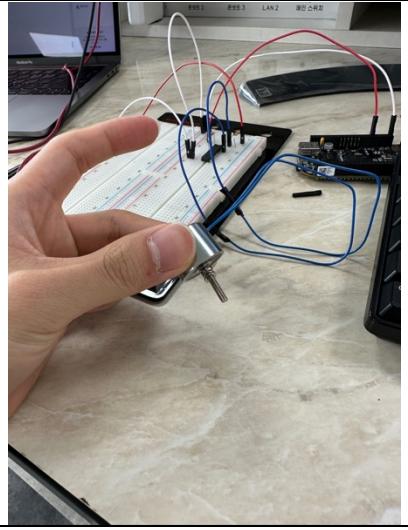
위의 코드들을 VSCode를 통해 CMakeLists.txt 파일을 수정하여 cross-compile하였다. gripper\_test라는 executable을 build하였다.

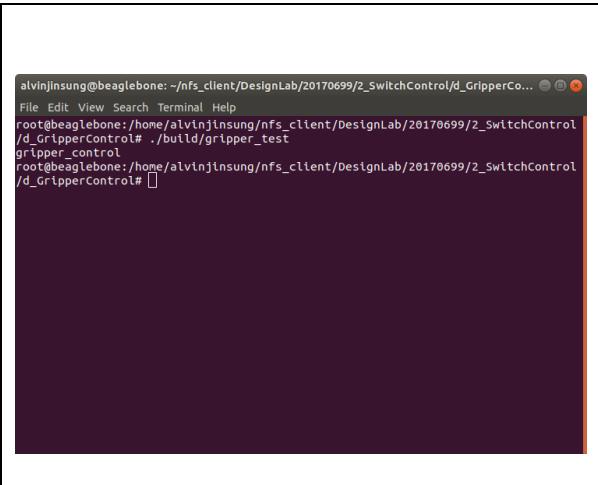
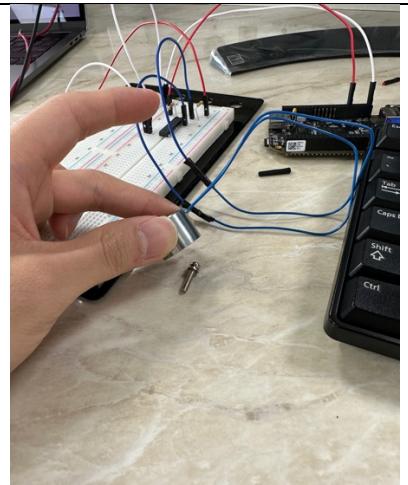
The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure under the folder `D_GRIPPERCONTROL`. It includes files like `CMakeLists.txt`, `gpio_control.hpp`, `gpio_control.cpp`, `gripper_control.hpp`, `gripper_control.cpp`, and `gripper_test.cpp`.
- Terminal:** Displays the command-line output of the cross-compilation process. The output shows the CMake configuration, the build command being executed (`/usr/bin/cmake --build /home/kaist/Desktop/20170699/2_SwitchControl/d_GripperControl/build --config Debug -j 8`), and the successful completion of the build with exit code 0.

그림 48. gripper\_test cross-compile 결과

Cross-compile이 성공적으로 완료되어 Beaglebone black에서 실행하며 solenoid magnet이 원하는 대로 잘 동작하는지 test해보았다. 결과는 아래와 같다.

 <pre>alvinjinsung@beaglebone: ~/nfs_client/DesignLab/20170699/2_SwitchControl/d_GripperCo... ●●● File Edit View Search Terminal Help root@beaglebone:~/nfs_client/DesignLab/20170699/2_SwitchControl/d_GripperControl# ./build/gripper_test gripper_control root@beaglebone:~/nfs_client/DesignLab/20170699/2_SwitchControl/d_GripperControl#</pre>	
그림 49. gripper_test 실행 중	그림 50. 실행 중에 solenoid magnet이 작동하는 모습

 <pre>alvinjinsung@beaglebone: ~/nfs_client/DesignLab/20170699/2_SwitchControl/d_GripperCo... ●●● File Edit View Search Terminal Help root@beaglebone:~/nfs_client/DesignLab/20170699/2_SwitchControl/d_GripperControl# ./build/gripper_test gripper_control root@beaglebone:~/nfs_client/DesignLab/20170699/2_SwitchControl/d_GripperControl#</pre>	
그림 51. gripper_test 실행 종료	그림 52. 지정된 시간이 지나고 solenoid magnet이 작동을 멈춘 모습

성공적으로 지정한 시간동안 solenoid magnet을 키고 끌 수 있었다.

#### 4. Discussion

(1) [Shell script vs. C program] Compare Shell script and C program to control LEDs. Please include the advantages and disadvantages of each on the report.

Shell script와 C program 모두 LED를 control 할 수 있으나 몇가지 차이점을 가지고 있다. Shell script는 shell에서 실행할 수 있는 command들로 이루어진 plain text file을 의미하는데, 쉽고 빠르게 작성할 수 있는데 반해 compile program보다는 느릴 수 있고 효율성이나 안정성이 떨어질 수 있다. C program은 이와 다르게 C언어로 작성된 코드를 compile하여 생성된 program을 의미한다. 이는 shell script보다는 작성하기 어렵고 복잡할 수 있으나 더 빠르고 효과적으로 동작을 수행할 수 있다. 조금 더 구체적으로 각각의 장단점을 설명하면 다음과 같다.

Shell script	
장점	단점
<ul style="list-style-type: none"> <li>- 쉽게 작성 가능하고 이해하기 쉽다.</li> <li>- 빠르게 prototype이 가능하다.</li> <li>- Compilation이나 linking이 필요 없다.</li> </ul>	<ul style="list-style-type: none"> <li>- Compiled program보다 느리다.</li> <li>- Compiled program에 비해 비효율적이고 안정성이 부족할 수 있다.</li> <li>- 복잡한 programming logic 구현이 어렵다.</li> </ul>

C program	
장점	단점
<ul style="list-style-type: none"> <li>- Shell script에 비해 효율적이고 빠르다.</li> <li>- Hardware를 직접 control하는데 사용할 수 있다.</li> <li>- 특정 hardware platform에 맡겨 최적화 할 수 있다.</li> <li>- 복잡한 programming logic 구현이 가능하다.</li> </ul>	<ul style="list-style-type: none"> <li>- Shell script에 비해 작성하고 이해하기 어렵다.</li> <li>- Compilation과 linking이 필요하다.</li> </ul>

(2) [Transistor array] Why do we require a transistor array to drive LEDs? Can you drive 3W LEDs using uLN2803?

LED를 작동하기 위해 transistor array가 많이 쓰이는데 이는 transistor array를 사용하면 여러개의 led를 하나의 component를 사용하여 쉽고 효율적으로 조작할 수 있기 때문이다. uLN2803의 경우 low-level signal을 switch하고 amplify 할 수 있는 Darlington pair가 내장되어 있어 lower current source를 가지고도 high-power led를 작동시킬 수 있다.

uLN2803을 이용해서 3W LED를 제대로 작동시킬 수 없다. uLN2803의 경우 각 채널별로 최대 500mA의 output current를 얻을 수 있는데, 이는 3W led를 작동 시키는데 필요한 700~1000mA에 미치지 못하는 수치이기 때문이다.

(3) [Solenoid magnet circuit] Today we applied external power supply to control the solenoid magnet. Write some basic idea to form an electronic circuit to control solenoid magnet when you apply voltage 1) using beaglebone's own GPIO signal and 2) using external power supply. Please include what you have to consider to apply stable electric power and control the solenoid magnet.

Electric circuit을 이용해 solenoid magnet을 control하기 위해서는 기본적인 필요 요소들이 존재 한다. 요소들은 아래와 같다.

- Solenoid magnet
- Driver circuit: solenoid magnet에 필요한 전류 및 전압 공급
- Control signal: Driver circuit을 control하고 solenoid를 키고 끄는데 필요한 signal
- Power supply: electric energy source

Driver circuit은 transistor나 MOSFET, relay등으로 구성할 수 있으며 control signal은 GPIO signal을 이용할 수 있다. Power supply로는 beaglebone 내장 power supply를 사용할수도, external power supply를 사용할 수도 있으며 external power supply의 경우 GPIO port와 external voltage source 사이의 voltage 차이를 반드시 유의하여야 한다.

Stable electric power를 위해서는 solenoid, driver circuit, GPIO등의 정격 전압과 전류를 정확히 확인하여 power supply나 저항값 등을 설정하여야 한다. Overhitting등을 방지하고 protection circuitry등을 마련하는 것 역시 중요하다.

**(4) [End effector] Solenoid magnet in this experiment will be used as a ‘gripper’ for a robotic manipulator project, in other words, an end effector. Please refer to the article follows, which explains about various kinds of the end effector.**

[<https://www.universal-robots.com/blog/robot-grippers-explained/>]

Choose one of the end effectors, and describe the differences in expected driving methods and relative advantage/disadvantages compared to solenoid magnet gripper.

Vacuum gripper는 internal gripper air pressure와 external air pressure의 차이를 이용해 물품을 들고, 이동시키는 end effector이다. Vacuum gripper는 suction cup, vacuum generator, control electronics로 이루어져 있다. Suction cup은 주로 flexible한 물질로 만들어져 물품에 잘 흡착하도록 제조되었다. Vacuum generator가 suction cup 내부를 vacuum 상태로 만들면 suction cup이 타겟으로 하는 물품에 흡착하여 물품을 잡고 고정할 수 있게 된다. Control electronics는 vacuum level을 결정하고 물품에 secure grip을 유지할 수 있도록 하는 역할을 한다.

균일한 모양과 크기의 물품에 적합한 solenoid magnet gripper와 비교하여 vacuum gripper는 다양한 모양과 크기의 물품을 다루기에 용이하다. 또한 solenoid magnet gripper가 강한 magnetic force로 인해 물품에 손상을 줄 수 있는데 반해 vacuum gripper는 조금 더 섬세하게 물품을 다룰 수 있다. 그러나 다공성이고 표면이 고르지 못한 물품의 경우 secure grip을 보장하기 어려울 수 있다. 또한 vacuum generator와 control exelectronics의 경우 조금 더 복잡하고 비싸 가격적인 면에서 경제성이 떨어질 수 있다.

**(5) [Discussion - own topic] Set your own topic to discuss related to Lab 2, and explain summary of your search result.**

Q. Describe the advantages and disadvantages of using GPIO compared to other communication protocols for controlling external devices?

A. GPIO는 간단하고 직관적인 communication protocol로 microcontroller나 single-board computer에 널리 활용되고 있다. Implement하기 쉬우며 additional hardware를 필요로 하지 않기에 simple project나 small-scale application에 효율적이다. 그러나 I2C, SPI등 다른 communication protocol과 비교해 한계점들도 존재한다. 우선 GPIO는 장거리 통신을 위해 디자인되지 않았다. 또한 high speed data transfer나 precise timing을 요하는 device를 control 하는데 용이하지 않을 수 있다. Pin의 개수도 유한하게 결정되어 있기에 multiple devices를 동시에 control하는데 효율적이지 못하다.

## 5. References

- [1] Lab2 Experiment Guide.pdf
- [2] Lab2 Procedure.pdf
- [3] EE405\_Lab2Lecture.pdf
- [4] EE405Lab2\_Video.mp4
- [5] The Shell Language vs. Other Programming Languages, <https://flylib.com/books/en/4.466.1.195/1/>
- [6] What's the difference between Scripting and Programming Languages?,  
<https://www.geeksforgeeks.org/whats-the-difference-between-scripting-and-programming-languages/>
- [7] ULN2803AN Datasheet,  
<https://pdf1.alldatasheet.com/datasheet-pdf/view/1159311/HMSEMI/ULN2803AN.html>
- [8] How to Drive High Power LEDs – 3W Aluminum Backed Star LEDs,  
<https://core-electronics.com.au/guides/how-to-drive-high-power-leds/>
- [9] ROBOT GRIPPERS EXPLAINED, <https://www.universal-robots.com/blog/robot-grippers-explained/>