[EE414] Embedded Systems

Lab2 Report

Woojin Lee
20180480
School of Computing, KAIST

April 7, 2021

1. Purpose

1. Purpose

Understand how to drive the GPIO LEDs in Beaglebone.
This experiment is for driving LED device on Beaglebone in user space. (not kernel space).

2.Problem Statement

LED Display
Write a program for LED display output on the embedded board, to output beats of the metronome visually using LED lamps with fixed tempo of 60 and fixed time-signature of 6/8.

2. Experiment Sequence

Step 1. Setup cross development environment
Step 2. Test LEDs with commands using sysfs
Step 3. Test LEDs with shell script
Step 4. Control LED with C and mmap
Step 5. Test Metronome*ledapplicationprogram*.

Step 0. Setup cross development environment for module

I will skip this part since we did it in Lab1.

Step 1. Test LEDs using sysfs and commands

1.1. Select LEDs for test

1.2. Check sysfs file

sysfs is one of virtual file system that linux kernel provides. You can get information about kernel system, HW information, etc...

I attatched output of series commands below.
Here, we can see that location of gpiochip32 is linked to other directory.
Also, -F command is for indicating type of contents. For example, @ for link, \for directory, etc...

```
woojin@beaglebone:~$ ls -F /sys/class/gpio
export    gpio117@  gpio27@  gpio38@  gpio50@  gpio69@  gpio78@  gpiochip0@
gpio10@   gpio12@   gpio3@   gpio39@  gpio51@  gpio7@   gpio79@  gpiochip32@
gpio11@   gpio13@   gpio30@  gpio4@   gpio60@  gpio70@  gpio8@   gpiochip64@
gpio110@  gpio14@   gpio31@  gpio44@  gpio61@  gpio71@  gpio80@  gpiochip96@
gpio111@  gpio15@   gpio32@  gpio45@  gpio62@  gpio72@  gpio81@  unexport
gpio112@  gpio2@    gpio33@  gpio46@  gpio63@  gpio73@  gpio86@
gpio113@  gpio20@   gpio34@  gpio47@  gpio65@  gpio74@  gpio87@
gpio114@  gpio22@   gpio35@  gpio48@  gpio66@  gpio75@  gpio88@
gpio115@  gpio23@   gpio36@  gpio49@  gpio67@  gpio76@  gpio89@
gpio116@  gpio26@   gpio37@  gpio5@   gpio68@  gpio77@  gpio9@
woojin@beaglebone:~$ cd /sys/class/gpio/gpiochip32
woojin@beaglebone:/sys/class/gpio/gpiochip32$ ls -F
base  device@  label  ngpio  power/  subsystem@  uevent
woojin@beaglebone:/sys/class/gpio/gpiochip32$ ls -al /sys/class/gpio/gpiochip32
lrwxrwxrwx 1 root gpio 0 Apr  2 08:25 /sys/class/gpio/gpiochip32 -> ../../devices/platform/
ocp/4804c000.gpio/gpio/gpiochip32
woojin@beaglebone:/sys/class/gpio/gpiochip32$
```

Figure 1: Output of commands

1.3. Find LED Sysfs

I find LED Sysfs, and here comes the output.

```
woojin@beaglebone:/sys/class/gpio/gpiochip32$ ls -F /sys/class | grep leds
leds/
woojin@beaglebone:/sys/class/gpio/gpiochip32$ ls -F /sys/class/leds
beaglebone:green:usr0@  beaglebone:green:usr2@
beaglebone:green:usr1@  beaglebone:green:usr3@
woojin@beaglebone:/sys/class/gpio/gpiochip32$
```

Figure 2: Output of commands

You can see the directoreis for controlling each of the user LEDs.

1.4. Get access right to usr0 LED

When I follow given commands, I can see many trigger modes, and current value is encapsulated with brackets. If I change trigger mode, LED blinks following the given manner.

1.5. Control on/off of usr0 LED

We can also control LED with brightness

1.6. Control periodic on/off of usr0 LED

If you set timer as trigger and give on and off time, you can control blinking pattern of LED.

Step 2. Test LEDs with shell script

Above, we changed LED behavior with sysfs interface. Here, we'll control LED with shell script.

2.1. Make a subdirectory b_gpio_led_shell

2.2. Edit control_led0.sh

Script Interpretation
If given argument is less then 2, print "Usage: control_led0.sh on_time off_time" and exit. Else, print "Control LED0 with on_time $1(first argument) and off_time $2(second argument)."
Then, provide arguments as we did in section 1.6.

2.3. Run shell script

I had to run with sudo option. If not, I got error message, "Permission denied".
I can see led working same as section 1.6.

Step 3. Control LED with C and mmap

3.1. Make a working directory

3.2. Build shell script to stop restore user LEDs

stop_user_leds.sh file interpretation
Almost same with Section 2. Makes usr0, usr1, usr2, usr3 LED trigger to "none" and prints output.
restore_user_leds.sh file interpretation
This makes to usr0 usr3 LED triggers to default value.

3.3. Test example program: pusLEDmmap.c with pushLEDmmap.h

Hardware

Pushbutton (additional device, not on the board)	ECAPPWM0 as GPIO_07
UART4 TXD as GPIO0_31	LED (additional device, not on the board)

About pushLEDmmap code
Push button(GPIO_07) will turn on LED(GPIO0_31)
However, in our experiment, it will not work because GPIO0_07, GPIO0_31 aren't connected to external devices like button, LED.
I also got some troubleshooting while executing pushLEDmmap. Thanks to some anonymous student in piazza, I could resolve the problem by executing with sudo command.
Below, I attached my Makefile for "make m1", pushLEDmmap.c and pushLEDmmap.h code with my additional comment, and caputred image about execution result.

```
woojin@beaglebone:~/nfs_client/lab2/pushLEDmmap$ sudo ./pushLEDmmap
Mapping 44E07000 - 44E08000 (size: 1000)
GPIO mapped to 0xb6f5d000
GPIO SETDATAOUTADDR mapped to 0xb6f5d194
GPIO CLEARDATAOUT mapped to 0xb6f5d190
Start copying GPIO_07 to GPIO_31
^C
Ctrl-C pressed, cleaning up and exiting...
woojin@beaglebone:~/nfs_client/lab2/pushLEDmmap$
```

Figure 3: pushLEDmmap execution result

Nothing special happens because GPIO0_07 and GPIO0_31 aren't connected to external devices.

Makefile

```
CC=arm-linux-gnueabihf-gcc

m1: pushLEDmmap.c pushLEDmmap.h
	$(CC) -o pushLEDmmap pushLEDmmap.c
```

comment on pushLEDmmap.h

```
// From: http://stackoverflow.com/questions/13124271/driving-beaglebone-gpio
// -through-dev-mem
// user contributions licensed under cc by-sa   3.0 with attribution required
// http://creativecommons.org/licenses/by-sa/3.0/
// http://blog.stackoverflow.com/2009/06/attribution-required/
// Author: madscientist159 (http://stackoverflow.com/users/3000377/madscientist159)

#ifndef _BEAGLEBONE_GPIO_H_
#define _BEAGLEBONE_GPIO_H_

// from AM335x Technical Reference Manual, pg 209
#define GPIO0_START_ADDR 0x44e07000
#define GPIO0_END_ADDR   0x44e08000
#define GPIO0_SIZE (GPIO0_END_ADDR - GPIO0_START_ADDR)

// from AM335x Technical Reference Manual, pg 211
#define GPIO1_START_ADDR 0x4804C000
#define GPIO1_END_ADDR   0x4804D000
#define GPIO1_SIZE (GPIO1_END_ADDR - GPIO1_START_ADDR)

// from AM335x Technical Reference Manual, pg 213
#define GPIO2_START_ADDR 0x41A4C000
#define GPIO2_END_ADDR   0x41A4D000
```

```
#define GPIO2_SIZE (GPIO2_END_ADDR - GPIO2_START_ADDR)

// from AM335x Technical Reference Manual, pg 213
#define GPIO3_START_ADDR 0x41A4E000
#define GPIO3_END_ADDR   0x41A4F000
#define GPIO3_SIZE (GPIO3_END_ADDR - GPIO3_START_ADDR)

// from our Lecture note 6, pg 13
#define GPIO_DATAIN 0x138
#define GPIO_SETDATAOUT 0x194
#define GPIO_CLEARDATAOUT 0x190

// for bit manipulation
#define GPIO_07  (1<<7)
#define GPIO_31  (1<<31)
#endif
```

comment on pushLEDmmap.c

```
// From: http://stackoverflow.com/questions/13124271/driving-beaglebone-gpio
// -through-dev-mem
// user contributions licensed under cc by-sa 3.0 with attribution required
// http://creativecommons.org/licenses/by-sa/3.0/
// http://blog.stackoverflow.com/2009/06/attribution-required/
// Author: madscientist159 (http://stackoverflow.com/users/3000377/madscientist159)
//
// Read one gpio pin and write it out to another using mmap.
// Be sure to set -O3 when compiling.

#include <stdio.h>
#include <stdlib.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <signal.h> // Defines signal-handling functions (i.e. trap Ctrl-C)
#include <unistd.h> // close()
#include "pushLEDmmap.h"

// Global variables
volatile int keepgoing = 1;

// Set to 0 when Ctrl-c is pressed
```

```
25 // Callback called when SIGINT is sent to the process (Ctrl-C)
26 void signal_handler(int sig) {
27     printf( "\nCtrl-C pressed, cleaning up and exiting...\n" );
28     keepgoing = 0;
29 }
30
31 int main(int argc, char *argv[]) {
32     volatile void *gpio_addr;
33     volatile unsigned int *gpio_datain;
34     volatile unsigned int *gpio_setdataout_addr;
35     volatile unsigned int *gpio_cleardataout_addr;
36
37     // Set the signal callback for Ctrl-C
38     // call signal_handler function when Ctrl-C comes in.
39     signal(SIGINT, signal_handler);
40
41     // file descripter for mmap
42     int fd = open("/dev/mem", O_RDWR);
43     printf("Mapping %X - %X (size: %X)\n", GPIO0_START_ADDR, GPIO0_END_ADDR, GPIO0_SIZE);
44
45     // mmap: memory mapping
46     // from GPIO0_START_ADDR, get GPIO0_SIZE amount of memory
47     gpio_addr = mmap(0, GPIO0_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, fd, GPIO0_START_ADDR);
48     gpio_datain = gpio_addr + GPIO_DATAIN;
49     gpio_setdataout_addr = gpio_addr + GPIO_SETDATAOUT;
50     gpio_cleardataout_addr = gpio_addr + GPIO_CLEARDATAOUT;
51
52     if(gpio_addr == MAP_FAILED) {
53         printf("Unable to map GPIO\n");
54         exit(1);
55     }
56
57     printf("GPIO mapped to %p\n", gpio_addr);
58     printf("GPIO SETDATAOUTADDR mapped to %p\n", gpio_setdataout_addr);
59     printf("GPIO CLEARDATAOUT mapped to %p\n", gpio_cleardataout_addr);
60     printf("Start copying GPIO_07 to GPIO_31\n");
61
62     while(keepgoing) {
63         // Since GPIO_07 is 1<<7, it only checks 7th bit.
64         if(*gpio_datain & GPIO_07) { //Check whether button is pushed
65             *gpio_setdataout_addr = GPIO_31;    //LED on
66         } else {
67             *gpio_cleardataout_addr = GPIO_31;  //LED off
68         }
```

```
69          //usleep(1);
70      }
71
72      // To remove mapping made by mmap function.
73      munmap((void *)gpio_addr, GPIO0_SIZE);
74      close(fd);
75
76      return 0;
77  }
```

3.4. Test prepared userLEDmmap.c

Here, I have to modify pushLEDmmap. pushLEDmmap deals with GPIO0_07, and GPIO0_31 which is not on board.
userLEDmmap uses user LEDs which is mounted on board.
Same as above, below, I attached my Makefile for "make m2", userLEDmmap.c and userLEDmmap.h code with my additional comment, and caputred image about execution result.
Also, I had to run stop_user_leds.sh file. At first, I didn't run it, and some LEDs were running in default pattern.
Makefile

```
1  CC=arm-linux-gnueabihf-gcc
2
3  m2: userLEDmmap.c userLEDmmap.h
4          $(CC) -o userLEDmmap userLEDmmap.c
```

userLEDmmap.h

```
1  // From: http://stackoverflow.com/questions/13124271/driving-beaglebone-gpio
2  // -through-dev-mem
3  // user contributions licensed under cc by-sa   3.0 with attribution required
4  // http://creativecommons.org/licenses/by-sa/3.0/
5  // http://blog.stackoverflow.com/2009/06/attribution-required/
6  // Author: madscientist159 (http://stackoverflow.com/users/3000377/madscientist159)
7
8  #ifndef _BEAGLEBONE_GPIO_H_
9  #define _BEAGLEBONE_GPIO_H_
10 #define GPIO0_START_ADDR 0x44e07000
11 #define GPIO0_END_ADDR   0x44e08000
12 #define GPIO0_SIZE (GPIO0_END_ADDR - GPIO0_START_ADDR)
13
```

```
14
15 // We will use GPIO1 since user LED is in GPIO1
16 #define GPIO1_START_ADDR 0x4804C000
17 #define GPIO1_END_ADDR   0x4804D000
18 #define GPIO1_SIZE (GPIO1_END_ADDR - GPIO1_START_ADDR)
19
20 #define GPIO2_START_ADDR 0x41A4C000
21 #define GPIO2_END_ADDR   0x41A4D000
22 #define GPIO2_SIZE (GPIO2_END_ADDR - GPIO2_START_ADDR)
23
24 #define GPIO3_START_ADDR 0x41A4E000
25 #define GPIO3_END_ADDR   0x41A4F000
26 #define GPIO3_SIZE (GPIO3_END_ADDR - GPIO3_START_ADDR)
27
28 #define GPIO_DATAIN 0x138
29 #define GPIO_SETDATAOUT 0x194
30 #define GPIO_CLEARDATAOUT 0x190
31
32 // I was bit confused, because beaglebone black manual says
33 // USR0, USR1, USR2, USR3 are allocated to GPIO1_21, GPIO2_22, GPIO2_23, GPIO2_24
34 // However, in my case, I followed lab document, GPIO1_21 ~ GPIO1_24, and it worked.
35 #define USER0   1<<21
36 #define USER1   1<<22
37 #define USER2   1<<23
38 #define USER3   1<<24
39 #endif
```

userLEDmmap.c

```
1  // From: http://stackoverflow.com/questions/13124271/driving-beaglebone-gpio
2  // -through-dev-mem
3  // user contributions licensed under cc by-sa 3.0 with attribution required
4  // http://creativecommons.org/licenses/by-sa/3.0/
5  // http://blog.stackoverflow.com/2009/06/attribution-required/
6  // Author: madscientist159 (http://stackoverflow.com/users/3000377/madscientist159)
7  //
8  // Read one gpio pin and write it out to another using mmap.
9  // Be sure to set -O3 when compiling.
10
11 // Change both .c and .h files to control four user LEDs, instead of the push button and the externa
12 // You may decide not to change the .h file.
13
14 // Check four user LEDs can be controlled (in an arbitrary manner).
```

```
15 // For example, you may try to blink the first user LED. In this step, just check the mapping works.
16
17 #include <stdio.h>
18 #include <stdlib.h>
19 #include <sys/mman.h>
20 #include <sys/types.h>
21 #include <sys/stat.h>
22 #include <fcntl.h>
23 #include <signal.h> // Defines signal-handling functions (i.e. trap Ctrl-C)
24 #include <unistd.h> // close()
25 #include "userLEDmmap.h"
26
27 // Global variables
28 volatile int keepgoing = 1;
29
30 // Set to 0 when Ctrl-c is pressed
31 // Callback called when SIGINT is sent to the process (Ctrl-C)
32 void signal_handler(int sig) {
33     printf( "\nCtrl-C pressed, cleaning up and exiting...\n" );
34     keepgoing = 0;
35 }
36
37 int main(int argc, char *argv[]) {
38     volatile void *gpio_addr;
39     volatile unsigned int *gpio_datain;
40     volatile unsigned int *gpio_setdataout_addr;
41     volatile unsigned int *gpio_cleardataout_addr;
42
43     // Set the signal callback for Ctrl-C
44     signal(SIGINT, signal_handler);
45
46     int fd = open("/dev/mem", O_RDWR);
47     printf("Mapping %X - %X (size: %X)\n", GPIO1_START_ADDR, GPIO1_END_ADDR, GPIO1_SIZE);
48
49     gpio_addr = mmap(0, GPIO1_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, fd, GPIO1_START_ADDR);
50     gpio_datain = gpio_addr + GPIO_DATAIN;
51     gpio_setdataout_addr = gpio_addr + GPIO_SETDATAOUT;
52     gpio_cleardataout_addr = gpio_addr + GPIO_CLEARDATAOUT;
53
54     if(gpio_addr == MAP_FAILED) {
55         printf("Unable to map GPIO\n");
56         exit(1);
57     }
58
```

```
59      printf("GPIO mapped to %p\n", gpio_addr);
60      printf("GPIO SETDATAOUTADDR mapped to %p\n", gpio_setdataout_addr);
61      printf("GPIO CLEARDATAOUT mapped to %p\n", gpio_cleardataout_addr);
62
63      printf("Manipulating LEDs\n");
64      int time = 1;
65      // pushLEDmmap uses usleep function, but usleep is in micro_second unit, so I modified it to sle
66      // turns on, turns off one by one
67      // should run stop_user_leds.sh first.
68      while(keepgoing) {
69          // turn on LEDs
70          printf("turning on LEDs\n");
71          *gpio_setdataout_addr = USER0;
72          sleep (time);
73          *gpio_setdataout_addr = USER1;
74          sleep (time);
75          *gpio_setdataout_addr = USER2;
76          sleep (time);
77          *gpio_setdataout_addr = USER3;
78          sleep (time);
79
80          // turn off LEDs
81          printf("turning off LEDs\n");
82          *gpio_cleardataout_addr = USER0;
83          sleep (time);
84          *gpio_cleardataout_addr = USER1;
85          sleep (time);
86          *gpio_cleardataout_addr = USER2;
87          sleep (time);
88          *gpio_cleardataout_addr = USER3;
89          sleep (time);
90      }
91
92      munmap((void *)gpio_addr, GPIO1_SIZE);
93      close(fd);
94
95      return 0;
96  }
```

```
woojin@beaglebone:~/nfs_client/lab2/userLEDmmap$ ls
Makefile  stop_user_leds.sh  userLEDmmap  userLEDmmap.c  userLEDmmap.h
woojin@beaglebone:~/nfs_client/lab2/userLEDmmap$ sudo ./userLEDmmap
Mapping 4804C000 - 4804D000 (size: 1000)
GPIO mapped to 0xb6f0e000
GPIO SETDATAOUTADDR mapped to 0xb6f0e194
GPIO CLEARDATAOUT mapped to 0xb6f0e190
Manipulating LEDs
turning on LEDs
turning off LEDs
turning on LEDs
turning off LEDs
turning on LEDs
turning off LEDs
turning on LEDs
turning off LEDs
turning on LEDs
turning off LEDs
^C
Ctrl-C pressed, cleaning up and exiting...
woojin@beaglebone:~/nfs_client/lab2/userLEDmmap$ ls
```

Figure 4: userLEDmmap output

Step 4. Test Metronome_led application program

4.1. Make a working directory

4.2. Edit prepared Metronome$_led.c$

I used same userLEDmmap.h file as header since we're utilizing same LEDs.

4.3. Make

4.4. Test on Beaglebone

I attached video file.

Makefile

```
CC=arm-linux-gnueabihf-gcc

m3: Metronome_led.c userLEDmmap.h
	$(CC) -o Metronome_led Metronome_led.c
```

userLEDmmap.h

```
// From: http://stackoverflow.com/questions/13124271/driving-beaglebone-gpio
// -through-dev-mem
// user contributions licensed under cc by-sa   3.0 with attribution required
// http://creativecommons.org/licenses/by-sa/3.0/
// http://blog.stackoverflow.com/2009/06/attribution-required/
// Author: madscientist159 (http://stackoverflow.com/users/3000377/madscientist159)

#ifndef _BEAGLEBONE_GPIO_H_
#define _BEAGLEBONE_GPIO_H_
```

```
#define GPIO0_START_ADDR 0x44e07000
#define GPIO0_END_ADDR   0x44e08000
#define GPIO0_SIZE (GPIO0_END_ADDR - GPIO0_START_ADDR)


// We will use GPIO1 since user LED is in GPIO1
#define GPIO1_START_ADDR 0x4804C000
#define GPIO1_END_ADDR   0x4804D000
#define GPIO1_SIZE (GPIO1_END_ADDR - GPIO1_START_ADDR)

#define GPIO2_START_ADDR 0x41A4C000
#define GPIO2_END_ADDR   0x41A4D000
#define GPIO2_SIZE (GPIO2_END_ADDR - GPIO2_START_ADDR)

#define GPIO3_START_ADDR 0x41A4E000
#define GPIO3_END_ADDR   0x41A4F000
#define GPIO3_SIZE (GPIO3_END_ADDR - GPIO3_START_ADDR)

#define GPIO_DATAIN 0x138
#define GPIO_SETDATAOUT 0x194
#define GPIO_CLEARDATAOUT 0x190

// I was bit confused, because beaglebone black manual says
// USR0, USR1, USR2, USR3 are allocated to GPIO1_21, GPIO2_22, GPIO2_23, GPIO2_24
// However, in my case, I followed lab document, GPIO1_21 ~ GPIO1_24, and it worked.
#define USER0   1<<21
#define USER1   1<<22
#define USER2   1<<23
#define USER3   1<<24
#endif
```

Metronome_led.c

```
// From: http://stackoverflow.com/questions/13124271/driving-beaglebone-gpio
// -through-dev-mem
// user contributions licensed under cc by-sa 3.0 with attribution required
// http://creativecommons.org/licenses/by-sa/3.0/
// http://blog.stackoverflow.com/2009/06/attribution-required/
// Author: madscientist159 (http://stackoverflow.com/users/3000377/madscientist159)
//
// Read one gpio pin and write it out to another using mmap.
// Be sure to set -O3 when compiling.

```

```c
// Change both .c and .h files to control four user LEDs, instead of the push button and the externa
// You may decide not to change the .h file.

// Check four user LEDs can be controlled (in an arbitrary manner).
// For example, you may try to blink the first user LED. In this step, just check the mapping works.

#include <stdio.h>
#include <stdlib.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <signal.h> // Defines signal-handling functions (i.e. trap Ctrl-C)
#include <unistd.h> // close()
#include "userLEDmmap.h"

// Global variables
volatile int keepgoing = 1;
unsigned int pattern[6] = {7, 1, 1, 3, 1, 1};

// Set to 0 when Ctrl-c is pressed
// Callback called when SIGINT is sent to the process (Ctrl-C)
void signal_handler(int sig) {
    printf( "\nCtrl-C pressed, cleaning up and exiting...\n" );
    keepgoing = 0;
}

int main(int argc, char *argv[]) {
    volatile void *gpio_addr;
    volatile unsigned int *gpio_datain;
    volatile unsigned int *gpio_setdataout_addr;
    volatile unsigned int *gpio_cleardataout_addr;

    // Set the signal callback for Ctrl-C
    signal(SIGINT, signal_handler);

    int fd = open("/dev/mem", O_RDWR);
    printf("Mapping %X - %X (size: %X)\n", GPIO1_START_ADDR, GPIO1_END_ADDR, GPIO1_SIZE);

    gpio_addr = mmap(0, GPIO1_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, fd, GPIO1_START_ADDR);
    gpio_datain = gpio_addr + GPIO_DATAIN;
    gpio_setdataout_addr = gpio_addr + GPIO_SETDATAOUT;
    gpio_cleardataout_addr = gpio_addr + GPIO_CLEARDATAOUT;

```

```
55      if(gpio_addr == MAP_FAILED) {
56          printf("Unable to map GPIO\n");
57          exit(1);
58      }
59
60      printf("GPIO mapped to %p\n", gpio_addr);
61      printf("GPIO SETDATAOUTADDR mapped to %p\n", gpio_setdataout_addr);
62      printf("GPIO CLEARDATAOUT mapped to %p\n", gpio_cleardataout_addr);
63
64      printf("Manipulating LEDs\n");
65      int period = 1;
66      int clk = 1e6 / period / 2; // every 0.5second
67
68      while(keepgoing) {
69          for(int i=0; i<6; i++){
70              printf("%d\n", pattern[i]);
71              *gpio_setdataout_addr = pattern[i] * USER0;
72              usleep(clk);
73              *gpio_cleardataout_addr = USER0 | USER1 | USER2 | USER3;
74              usleep(clk);
75          }
76      }
77
78      munmap((void *)gpio_addr, GPIO1_SIZE);
79      close(fd);
80
81      return 0;
82  }
```

I used 7, 3, 1, so that I can manipulate multiple bits in once.
Also, I set usleep function again, because I have to change the LED status in milisecond scale.

3. Experimental Results

I attached demonstration video, so you can refer that that one.

```
Stop user LED1
Stop user LED2
Stop user LED3
woojin@beaglebone:~/nfs_client/lab2/k_metro_user_leds$ sudo ./Metronome_led
Mapping 4804C000 - 4804D000 (size: 1000)
GPIO mapped to 0xb6fb9000
GPIO SETDATAOUTADDR mapped to 0xb6fb9194
GPIO CLEARDATAOUT mapped to 0xb6fb9190
Manipulating LEDs
7
1
1
3
1
^C
Ctrl-C pressed, cleaning up and exiting...
1
sh jin@beaglebone:~/nfs_client/lab2/k_metro_user_leds$ sudo ./restore_user_leds.s
./restore_user_leds.sh: line 8: echo: write error: Invalid argument
Restore user LED0
Restore user LED1
Restore user LED2
Restore user LED3
woojin@beaglebone:~/nfs_client/lab2/k_metro_user_leds$
```

Figure 5: Metronome output

4. Discussion

Question 1

We have two methods for LED drive:

1) A method using sysfs in shell script.

2) A method using mmap in application.

Compare these two methods. Describe advantages and disadvantages of these methods.

Acutally, sysfs is much more simple than mmap. However, you cannot elaborately control LED with sysfs. Also, both sysfs and mmap may occur permission problem. These two methods run at the user space, so it's simpler than device driver module that we're going to discuss at question2, while having limited access to hardware and memory.

Question 2

Since user LED is a device, it should be controlled by device driver module in kernel level. Discuss advantages and disadvantages of this method.

As I mentioned above, sysfs and mmap may occur permission problem. For example, when there exist multiple users. If you use device driver, you can control these points as you want. However, device driver is the most complicated method. Also, module runs at the kernel space, and invoked by either system call or hardware interrupt. So it can have full access to hardware and memory.

5. References

[1] Beaglebone System Reference manual A6.

Beaglebone Black System Reference manual A5.2

AM335x ARM Cortex-A8 Microprocessors (MPUs) Technical Reference Manual (Rev.

F), Texas Instruments,
AM335x ARM Cortex-A8 Microprocessors (MPUs) (Rev. D) , Texas Instruments
Lab2 document