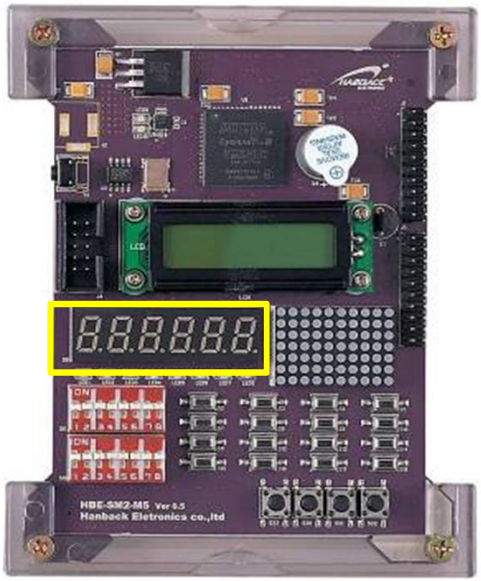
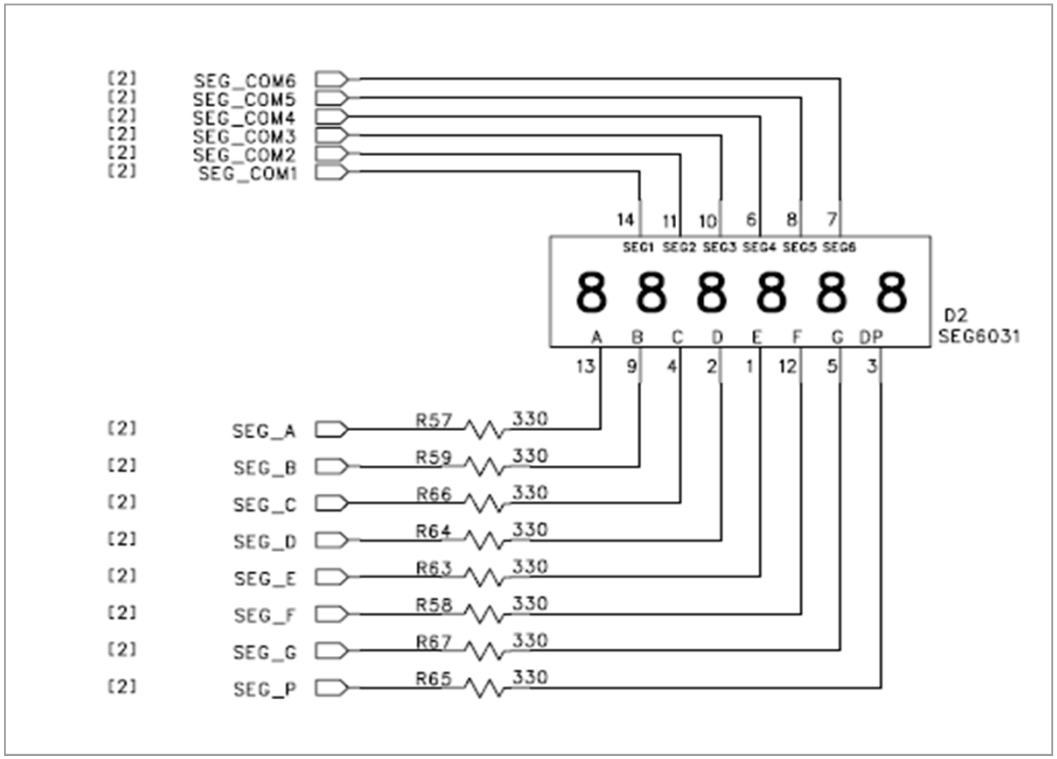
KWANGWOON UNIVERSITY

Device driver example
- 7 segment -

H/W user manual for 7-segment

- **6자리의 7-segment**
  - Digit Register: to identify each grid.
  - Data Register: to output the segment data





H/W user manual for 7-segment

- **7-segment LED Digit Register**

Physical Address : 0x1480_0000

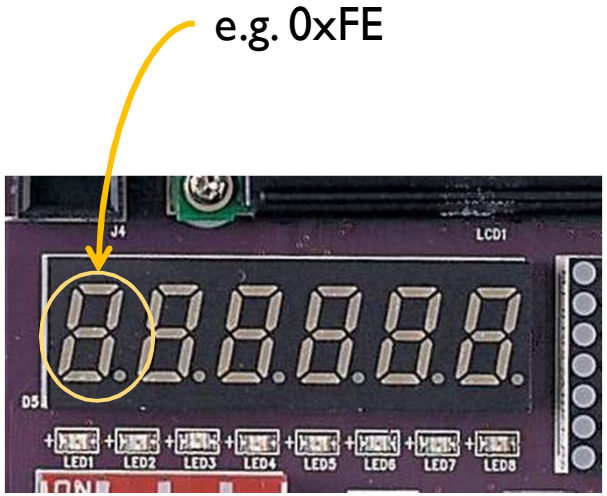
7 Segment LED Digit Register Peripheral Registers

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	X	X	X	X	X	X	X	X	X	X	COM6	COM5	COM4	COM3	COM2	COM1
Reset	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Bits	Name	Description
0	COM1	"SEG_COM1" Digit Segment Select Bit(Active Low)
1	COM2	"SEG_COM2" Digit Segment Select Bit(Active Low)
2	COM3	"SEG_COM3" Digit Segment Select Bit(Active Low)
3	COM4	"SEG_COM4" Digit Segment Select Bit(Active Low)
4	COM5	"SEG_COM5" Digit Segment Select Bit(Active Low)
5	COM6	"SEG_COM6" Digit Segment Select Bit(Active Low)

H/W user manual for 7-segment

- **Grid selection**
  - 0xFE – SEG1 (1111 1110)
  - 0xFD – SEG2 (1111 1101)
  - 0xFB – SEG3 (1111 1011)
  - 0xF7 – SEG4 (1111 0111)
  - 0xEF – SEG5 (1110 1111)
  - 0xDF – SEG6 (1101 1111)



H/W user manual for 7-segment

- **7-segment LED Data Register**

Physical Address : 0x1480_1000

7 Segment LED Data Register

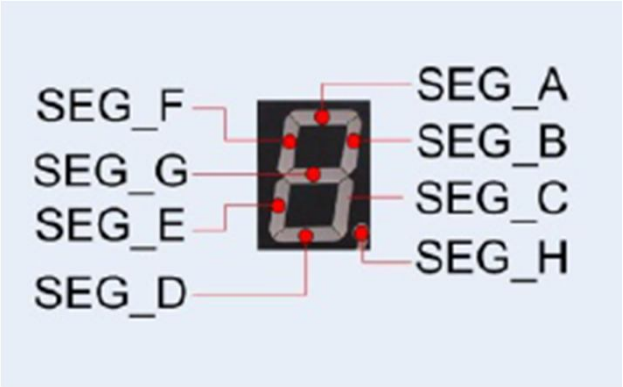
Peripheral Registers

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	X	X	X	X	X	X	X	X	SEG_H	SEG_G	SEG_F	SEG_E	SEG_D	SEG_C	SEG_B	SEG_A
Reset	x	x	x	x	x	X	x	x	x	x	x	x	x	x	x	x

Bits	Name	Description
0	SEG_A	'A' Segment Select Bit
1	SEG_B	'B' Segment Select Bit
2	SEG_C	'C' Segment Select Bit
3	SEG_D	'D' Segment Select Bit
4	SEG_E	'E' Segment Select Bit
5	SEG_F	'F' Segment Select Bit
6	SEG_G	'G' Segment Select Bit
7	SEG_H	'H' Segment Select Bit

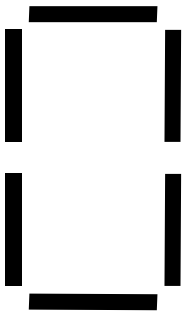
H/W user manual for 7-segment

- **Segment Data**
  - 0x3f – '0' (0011 1111)
  - 0x06 – '1' (0000 0110)
  - 0x5b – '2' (0101 1011)
  - 0x4f – '3' (0100 1111)
  - 0x66 – '4' (0110 0110)
  - 0x6d – '5' (0110 1101)
  - 0x7d – '6' (0111 1101)
  - 0x07 – '7' (0000 0111)
  - 0x7f – '8' (0111 1111)
  - 0x6f – '9' (0110 1111)

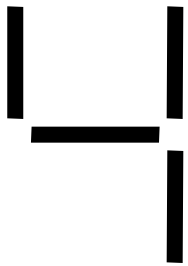


<meaning of each bit>

e.g. 0x3f ➔

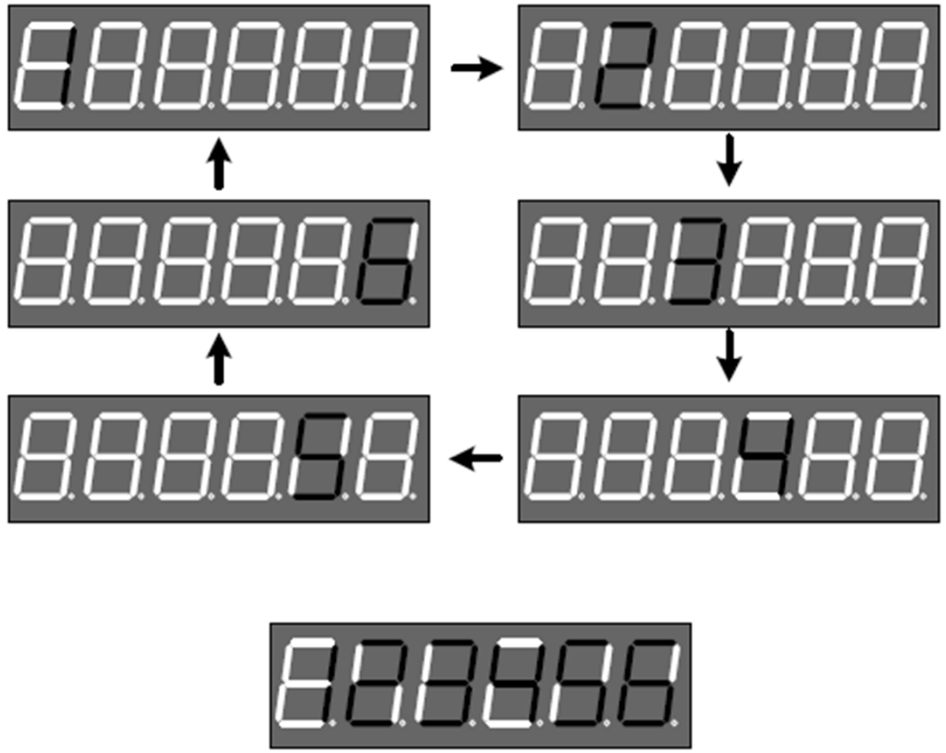


e.g. 0x66 ➔



H/W user manual for 7-segment

- **E.g. “123456”**
  - Set “0x06” to data register & set “0xFE” to digit register
  - Set “0x5b” to data register & set “0xFD” to digit register
  - …



7-segment control using mmap()

```
/* 7segment.c */
#include <stdlib.h>
#include <unistd.h>
#include <sys/mman.h>
#include <asm/fcntl.h>

unsigned char Getsegcode(unsigned char x);

#define SEGMENT1    0x14800000
#define SEGMENT2    0x14801000

int main(char argc, char **argv)
{
    unsigned char *addr_seg1, *addr_seg2;
    char data[6];
    char digit[6]={0x20, 0x10, 0x08, 0x04, 0x02, 0x01};
    int fd,i,j,k;
    int count=0, temp1,temp2,value;

    if(argc == 1) {
        value = 50;
    } else if (argc == 2) {
        value =  strtol(&argv[1][0],NULL ,10);
        if(value > 100 ) value = 100;
    } else {
        printf("please input the parameter![1 - 100] ex) ./7segment 10 \n");
    }
```

7-segment control using mmap()

```
if((fd=open("/dev/7segment",O_RDWR|O_SYNC)) < 0) {
    perror(“device file open is failed\n");
    exit(1);
}

addr_seg1= (unsigned char *)mmap(NULL, 4096,PROT_WRITE, MAP_SHARED, fd, SEGMENT1);

addr_seg2= (unsigned char *)mmap(NULL, 4096,PROT_WRITE, MAP_SHARED, fd, SEGMENT2);

if(addr_seg1 < 0 || addr_seg2 < 0) {
    close(fd);
    perror("mmap error\n");
    exit(1);
}
```

7-segment control using mmap()

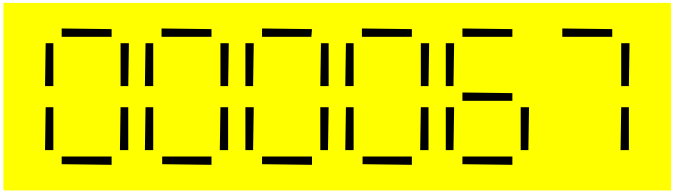
```
    while(count<value) {

        data[5]=Getsegcode(count/100000);
        temp1=count%100000;
        data[4]=Getsegcode(temp1/10000);
        temp2=temp1%10000;
        data[3]=Getsegcode(temp2/1000);
        temp1=temp2%1000;
        data[2]=Getsegcode(temp1/100);
        temp2=temp1%100;
        data[1]=Getsegcode(temp2/10);
        data[0]=Getsegcode(temp2%10);

        for(j=0;j<50;j++) {
            for(i=0;i<6;i++) {
                *addr_seg1 = ~digit[i];
                *addr_seg2 = data[i];
                for(k=0;k<65536;k++);
            }
        }
        count++;
    }

    usleep(1000);
    munmap(addr_seg1,4096);
    munmap(addr_seg2,4096);
    close(fd);
    return 0;
}
```

e.g. if (count == 67)



7-segment control using mmap()

```
unsigned char Getsegcode(unsigned char x)
{
    unsigned char code;

    switch (x) {
        case 0 : code = 0x3f;       break;
        case 1 : code = 0x06;       break;
        case 2 : code = 0x5b;       break;
        case 3 : code = 0x4f;       break;
        case 4 : code = 0x66;       break;
        case 5 : code = 0x6d;       break;
        case 6 : code = 0x7d;       break;
        case 7 : code = 0x07;       break;
        case 8 : code = 0x7f;       break;
        case 9 : code = 0x6f;       break;
        default : code = 0;         break;
    }
    return code;
}
```

7-segment device driver

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/fs.h>
#include <linux/types.h>
#include <linux/ioport.h>
#include <linux/errno.h>
#include <linux/delay.h>

#include <asm/io.h>
#include <asm/ioctl.h>
#include <asm/hardware.h>
#include <asm/uaccess.h>
#include <asm/fcntl.h>

#define DRIVER_AUTHOR                   "Hanback Electronics"
#define DRIVER_DESC                     "7 Segment test"
#define SEGMENT_MODULE_VERSION          "7 Segment PORT V0.1"
#define SEGMENT_NAME                    "7 Segment"
#define SEGMENT_MAJOR_NUMBER            0

#define SEGMENT_ADDRESS                 0x14800000
#define SEGMENT_ADDRESS_RANGE           0x2000

//Global variable
static unsigned int segment_usage = 0;
static unsigned int segment_major = 0;
static unsigned int segment_ioremap;
static unsigned char *segment_data;
static unsigned char *segment_grid;
```

7-segment device driver

```
int segment_open (struct inode *inode, struct file *filp)
{
    if(segment_usage != 0) return -EBUSY;

    segment_ioremap = (unsigned int)ioremap(SEGMENT_ADDRESS, SEGMENT_ADDRESS_RANGE);

    if(!check_mem_region(segment_ioremap,SEGMENT_ADDRESS_RANGE))
            request_region(segment_ioremap, SEGMENT_ADDRESS_RANGE, SEGMENT_NAME);
    else
            printk(KERN_WARNING"Can't get IO Region 0x%x\n", segment_ioremap);

    segment_grid = (unsigned char *)(segment_ioremap);
    segment_data = (unsigned char *)(segment_ioremap + 0x1000);

    segment_usage = 1;
    return 0;
}

int segment_release (struct inode *inode, struct file *filp)
{
    release_region(segment_ioremap, SEGMENT_ADDRESS_RANGE);
    iounmap((unsigned char *)segment_ioremap);
    segment_usage = 0;
    return 0;
}
```

7-segment device driver

```
unsigned char Getsegmentcode (char x)
{
    unsigned char code;
    switch (x) {
        case 0x0 : code = 0x3f; break;
        case 0x1 : code = 0x06; break;
        case 0x2 : code = 0x5b; break;
        case 0x3 : code = 0x4f; break;
        case 0x4 : code = 0x66; break;
        case 0x5 : code = 0x6d; break;
        case 0x6 : code = 0x7d; break;
        case 0x7 : code = 0x07; break;
        case 0x8 : code = 0x7f; break;
        case 0x9 : code = 0x6f; break;
        case 0xA : code = 0x77; break;
        case 0xB : code = 0x7c; break;
        case 0xC : code = 0x39; break;
        case 0xD : code = 0x5e; break;
        case 0xE : code = 0x79; break;
        case 0xF : code = 0x71; break;
        default : code = 0; break;
    }
    return code;
}
```

```
A : 0111 0111 : A
B : 0111 1100 : b
C : 0011 1001 : C
D : 0101 1110 : d
E : 0111 1001 : E
F : 0111 0001 : F
```

7-segment device driver

```
ssize_t segment_write(struct file *inode, const char *gdata, size_t length, loff_t *off_what)
{
    unsigned char data[6];
    unsigned char digit[6]={0x20, 0x10, 0x08, 0x04, 0x02, 0x01};
    unsigned int i,j,num,ret;
    unsigned int count=0,temp1,temp2;

    ret = copy_from_user(&num,gdata,4);
    if (ret < 0) return -1;

    while(count< num+1) {
        data[5]=Getsegmentcode(count/100000);
        temp1=count%100000;
        data[4]=Getsegmentcode(temp1/10000);
        temp2=temp1%10000;
        data[3]=Getsegmentcode(temp2/1000);
        temp1=temp2%1000;
        data[2]=Getsegmentcode(temp1/100);
        temp2=temp1%100;
        data[1]=Getsegmentcode(temp2/10);
        data[0]=Getsegmentcode(temp2%10);
        for(j=0;j<100;j++) {
            for(i=0;i<6;i++) {
                *segment_grid = ~digit[i];
                *segment_data = data[i];
                mdelay(1);
            }
        }
        count++;
    }
    return length;
}
```

copy_from_user(to, from, n)
- copy a block of data from user space
  to: destination address, in kernel space.
  from: source address, in user space.
  n: number of bytes to copy.

7-segment device driver

```
struct file_operations segment_fops =
{
    .owner              = THIS_MODULE,
    .open               = segment_open,
    .write              = segment_write,
    .release            = segment_release,

};
```

7-segment device driver

```
int segment_init(void)
{
    int result;

    result = register_chrdev(SEGMENT_MAJOR_NUMBER, SEGMENT_NAME, &segment_fops);
    if (result < 0) {
        printk(KERN_WARNING"Can't get any major\n");
        return result;
    }

    segment_major = result;
    printk("init module, 7segment major number : %d\n", result);
    return 0;
}

void segment_exit(void)
{

      unregister_chrdev(segment_major,SEGMENT_NAME);

}

module_init(segment_init);
module_exit(segment_exit);

MODULE_AUTHOR(DRIVER_AUTHOR);
MODULE_DESCRIPTION(DRIVER_DESC);
MODULE_LICENSE("GPL");
```

KWANGWOON UNIVERSITY

Application for 7-segment driver test

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>


int main()
{

    int fd;
    int value;

        if((fd=open("/dev/7segment",O_RDWR|O_SYNC)) < 0) {
                printf("FND open fail\n");
                exit(1);
        }

        while(value != 0) {
                printf("Input counter value : (0 : exit program) \n");
                scanf("%d", &value);
                write(fd,&value,4);
        }

        close(fd);
        return 0;

}
```