결과 보고서

|  |  |  |  |
| --- | --- | --- | --- |
| **학부** | 소프트웨어학과 | **실습실** | 318호 |
| **이름** | 201120910 | **학번** | 201120890 |
| 김민우 | 김얼 |
| **결과 화면** |  | | |
| **구현사항** | * 4x4 Key-Matrix의 key input을 line by line으로 출력. * 좌 상단 (1.1) 부터 우 하단 (4.4)로 표현. | | |
| **미 구현**  **사항** | * While(1) 내에서 첫 read()부터 다음 read()까지 보드의 입력차이를 usleep()으로 대체. | | |

|  |  |
| --- | --- |
| **소스코드** | -KeyMxApp.c- (User Application)  #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <sys/ioctl.h>  #include <ctype.h>  #include <sys/ipc.h>  #include <sys/types.h>  #include <sys/stat.h>  #include <fcntl.h>  #include <unistd.h>  int main(int argc, char \*\*argv){  int rdata;  int fd;  int x, y;  while(1){  rdata = 0;  fd = open("/dev/keymx", O\_RDWR);    if (fd <0){  perror("driver open error.\n");  return 1;  }    while (1) {  usleep(150000);  read(fd, &rdata, 4);  if (rdata != 0) break;  }    x = 0;  y = 0;    x = rdata / 4;  y = rdata % 4;  if(y == 0){  x -=1;  y =4;  }  printf("button : (%d,%d)\n",x+1,y);    close(fd);  }  return rdata;  } |
| **소스코드** | -Keymxdrv.c- (Kernel Module Driver)  #include <linux/miscdevice.h>  #include <linux/fs.h>  #include <linux/kernel.h>  #include <linux/module.h>  #include <asm/io.h>  #include <asm/uaccess.h>  #include <linux/init.h>  #include <linux/platform\_device.h>  #define KEY\_SW\_ADDRESS (0x06000000 + 0x4000)  #define ADDRESS\_MAP\_SIZE 0x1000  volatile unsigned short \*keymx\_base;  ssize\_t keymx\_read(struct file \*pfile, char \*buf, size\_t count, loff\_t \*filePos){  unsigned short rdata;  unsigned int ret;  int bitn = 0;  int i,j;  for(i=0;i<4;i++){  writew(0x10<<i, keymx\_base);  rdata = readw(keymx\_base);  if(rdata > 0){  for(j=0;j<4;j++){  if(0x01<<j & rdata){  bitn = i + 1 + j\*4;  ret = copy\_to\_user(buf,&bitn,4);  }  }  }  }  return 4;  }  int keymx\_open(struct inode \*inode, struct file \*pfile){  if(check\_mem\_region(KEY\_SW\_ADDRESS, ADDRESS\_MAP\_SIZE)){  printk("keyswdrv : memory already in use\n");  return -EBUSY;  }  if(request\_mem\_region(KEY\_SW\_ADDRESS, ADDRESS\_MAP\_SIZE, "KEYMX") == NULL){  printk("keyswdrv : fail to allocate mem region\n");  return -EBUSY;  }  keymx\_base = ioremap(KEY\_SW\_ADDRESS, ADDRESS\_MAP\_SIZE);  if(keymx\_base == NULL){  printk("keyswdrv : fail to io mapping\n");  release\_mem\_region(KEY\_SW\_ADDRESS, ADDRESS\_MAP\_SIZE);  return -EBUSY;  }  return 0;  }  int keymx\_release(){  if(keymx\_base != NULL){  iounmap(keymx\_base);  keymx\_base = NULL;  release\_mem\_region(KEY\_SW\_ADDRESS, ADDRESS\_MAP\_SIZE);  }  return 0;  }  struct file\_operations keymx\_fops = {  .owner = THIS\_MODULE,  .open = keymx\_open,  .read = keymx\_read,  .release = keymx\_release,  };  struct miscdevice keymx\_device={  .minor = MISC\_DYNAMIC\_MINOR,  .name = "keymx",  .fops = &keymx\_fops,  };  static int \_\_init keymx\_init(void){  int res;    res = misc\_register(&keymx\_device);  if(res){  printk("fail to register the device\n");  return res;  }  return 0;  }  static void \_\_exit keymx\_exit(void){  misc\_deregister(&keymx\_device);  }  module\_init(keymx\_init)  module\_exit(keymx\_exit) |