7주차

1. Interrupt vs. Polling (최종)

지난주 마무리 짓지 못하였던 interrupt와 polling의 비교를 마무리 지었다. 먼저, interrupt의 경우 lock을 사용하려 하였는데, 이 계획대로 하지 않고 SW interrupt인 Signal을 사용하였다. 다음은 interrupt용 linux device driver이다.

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/gpio.h>
#include <linux/interrupt.h>
#include <linux/io.h>
#include <linux/device.h>
#include <linux/delay.h>
#include <linux/unistd.h>
#include <linux/cdev.h>
#include <linux/uaccess.h>
#include <linux/fs.h>
#include <linux/sched.h>
#define TIMER_ADDR_BASE 0xA0000000
#define TIMER_ADDR_HIGH 0xA000FFFF
#define TIMER_DOWN_COUNT_OFFSET 0x00000002
#define TIMER_LOAD_REG_OFFSET 0x00000004
#define TIMER_VALUE 0x000186A0
#define TIMER_MASK_AUTO_RELOAD 0x00000010
#define TIMER_MASK_LOAD_ENABLE 0x00000020
#define TIMER_MASK_INTR_ENABLE 0x00000040
#define TIMER_MASK_START 0x00000080
#define TIMER_MASK_CLEAR_INTR 0x00000100
#define DEVICE_NAME "timer"
#define MINOR_BASE 0
static unsigned int IRQNum;
static dev_t my_dev;
static struct cdev my_cdev;
static struct class *my_class;
static unsigned int irqnum = 49; // or 52? Finally, 49
static int process_pid;
unsigned int prev;
void* timer_addr;
struct task_struct *my_task;
static irq_handler_t irq_handler(unsigned int irq, void *dev_id, struct pt_regs *regs){
    //prev = ioread32(timer_addr);
    //iowrite32(prev|TIMER_MASK_CLEAR_INTR,timer_addr);
    printk(KERN_ALERT "Timer interrupted! \n");
    send_sig(SIGUSR1,my_task,0);
    printk(KERN_ALERT "Signal was sent from Kernel! \n");
    return (irq_handler_t) IRQ_HANDLED;
}
static ssize_t timer_write(struct file *filp, const char __user *buf, size_t count,
loff_t *f_pos){
    int ret;
    unsigned int val;
```

```
    if(count != sizeof(unsigned int)){
        printk(KERN_ALERT "Invalid size of data\n");
        return -EINVAL;
    }
    ret = copy_from_user(&val, buf, count);
    if(ret < 0){
        printk(KERN_ALERT "copy_from_user failed\n");
        return -EFAULT;
    }
    printk(KERN_INFO "Process ID is %d\n", val);
    process_pid = val;
    my_task = pid_task(find_vpid(process_pid), PIDTYPE_PID);
    prev = ioread32(timer_addr);
    iowrite32(prev|TIMER_MASK_START,timer_addr);
    return count;
}
static struct file_operations my_fops = {
    .owner = THIS_MODULE,
    .write = timer_write,
};
static int timer_init(void){
    /* Device driver Registration */
    int ret;
    ret = alloc_chrdev_region(&my_dev, MINOR_BASE, 1, DEVICE_NAME);
    if(ret < 0){
        printk(KERN_ERR "alloc_chrdev_region failed\n");
        return ret;
    }
    cdev_init(&my_cdev, &my_fops);
    ret = cdev_add(&my_cdev, my_dev, 1);
    if(ret < 0){
        printk(KERN_ERR "cdev_add failed\n");
        return ret;
    }
    my_class = class_create(THIS_MODULE, DEVICE_NAME);
    if(IS_ERR(my_class)){
        printk(KERN_ERR "class_create failed\n");
        return PTR_ERR(my_class);
    }
    device_create(my_class, NULL, my_dev, NULL, DEVICE_NAME);
    request_irq(irqnum,        (irq_handler_t)irq_handler,        IRQF_TRIGGER_RISING,
"timer_interrupt", NULL);
    /* Timer Setting */
    timer_addr  = ioremap(TIMER_ADDR_BASE,TIMER_ADDR_HIGH-TIMER_ADDR_BASE+1);
    if(timer_addr == NULL){
        printk(KERN_ERR "ioremap failed\n");
        return -1;
    }
    iowrite32(TIMER_VALUE,timer_addr+TIMER_LOAD_REG_OFFSET);
    /*Timer value setting*/
    iowrite32(TIMER_MASK_LOAD_ENABLE,timer_addr);
    /*Timer load enable, it should be clear.*/
    iowrite32(TIMER_MASK_INTR_ENABLE         |         TIMER_MASK_AUTO_RELOAD         |
TIMER_DOWN_COUNT_OFFSET,timer_addr);


    /*Timer interrupt enable & auto reload, then timer start.*/
    return 0;
```

```
}
static void timer_exit(void){
    //iowrite32(0,timer_addr);
    free_irq(irqnum, NULL);
    iounmap(timer_addr);
    device_destroy(my_class, my_dev);
    class_destroy(my_class);
    cdev_del(&my_cdev);
    unregister_chrdev_region(my_dev, 1);
}
module_init(timer_init);
module_exit(timer_exit);
MODULE_LICENSE("GPL");
```

Interrupt handler의 경우 한번 만 interrupt를 처리하면 되기 때문에 AXI Timer의 interrupt clear는 주석처리를 하였고, handler가 호출되었다는 것은 1ms가 경과되어 계산이 완료되었다는 뜻으로 user에게 계산이 완료되었다는 signal을 보낸다. Signal은 linux상의 SIGUSR1을 사용하였다.

추가로 User에서 엔터를 누르면 write가 실행되고, kernel에서는 write 함수 내에서 timer를 시작한다. Clock은 100MHZ이므로 1ms를 측정하기 위해서는 Timer load register에 100000, 16진수로 0x000186A0를 넣어준다. Signal을 kernel에서 해당 user process로 보내주기 위해 write시 PID를 넘겨 받아 static 전역 변수에 저장해두고, interrupt handler에서 이를 사용한다.

추가로, 지난주 polling 결과에서 약 1초에 해당하는 수치들이 나왔는데, 코드를 살펴보니 애초에 sleep(1)을 하고 read를 하여 의미가 없는 행동임을 파악하여, 주기가 없는 busy-wait, 1ms의 busy-wait 두 가지를 측정하여 다시 결과를 내었다.

아래는 User application code이다.

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <time.h>
#include <signal.h>
#define BASE_ADDR_BRAM_1 0xA0010000
#define BASE_ADDR_BRAM_2 0xA0012000
#define LEN 8
void *bram_addr_1;
void *bram_addr_2;
void handler(){
    memcpy(bram_addr_1,bram_addr_2,LEN);
    printf("Signal Received\n");
}
```

```c
int main(){
    pid_t pid;
    clock_t start_1,end_1,start_2,end_2;
    sigset_t set;
    struct sigaction act;
    int signum;
    char input;
    int fd, fd_2, fd_3;
    pid = getpid();
    printf("This process ID is %d\n",pid);
    fd = open("/dev/mem",O_RDWR);
    fd_2 = open("/dev/mem",O_RDWR);
    fd_3 = open("/dev/timer",O_RDWR);
    bram_addr_1 = mmap(0,LEN,PROT_READ|PROT_WRITE,MAP_SHARED,fd,BASE_ADDR_BRAM_1);
    bram_addr_2 = mmap(0,LEN,PROT_READ|PROT_WRITE,MAP_SHARED,fd_2,BASE_ADDR_BRAM_2);
    if(bram_addr_1 == MAP_FAILED || bram_addr_2 == MAP_FAILED){
        if(bram_addr_1 != MAP_FAILED){
            perror("Mapping Failed : BRAM_2\n");
            close(fd);
        }
        else if(bram_addr_2 != MAP_FAILED){
            perror("Mapping Failed : BRAM_1\n");
            close(fd_2);
        }
        else{
            perror("Mapping Failed : Both\n");
        }
        exit(1);
    }
    printf("Mapping Successful\n");
    start_1 = clock();
    memset(bram_addr_1,0,LEN);
    end_1 = clock();
    printf("First Phase takes %f seconds\n",(double)(end_1 - start_1)/CLOCKS_PER_SEC);
    sigemptyset(&set);
    sigaddset(&set,SIGUSR1);
    act.sa_handler = handler;
    act.sa_flags = 0;
    sigaction(SIGUSR1,&act,NULL);
    scanf("%c",&input);
    start_2 = clock();
    write(fd_3,&pid,sizeof(pid));
    sigwait(&set,&signum);
    end_2 = clock();
    printf("Second Phase takes %f seconds\n",(double)(end_2 - start_2)/CLOCKS_PER_SEC);
    munmmap(bram_addr_1,LEN);
    munmmap(bram_addr_2,LEN);
    close(fd);
    close(fd_2);
    close(fd_3);
    return 0;
}
```

결과는 아래와 같다.

Polling (1s delay)			
	A phase (sec)	B phase (sec)	Total (sec)
1	0.000007	1.006374	1.006381
2	0.000008	1.004457	1.004465
3	0.000018	0.972174	0.972192
4	0.00001	0.998068	0.998078
5	0.000008	0.992523	0.992531
6	0.000008	1.015169	1.015177
7	0.000008	1.008677	1.008685
8	0.000018	1.004888	1.004906
9	0.000009	1.013305	1.013314
10	0.000008	1.014039	1.014047
AVG	0.0000102	1.0029674	1.002978
MAX	0.000018	1.015169	1.015177
MIN	0.000007	0.972174	0.972192
MEDIAN	0.000008	1.005631	1.005644

Interrupt			
	A phase (sec)	B phase (sec)	Total (sec)
1	0.000005	0.003008	0.003013
2	0.000006	0.002995	0.003001
3	0.000005	0.002993	0.002998
4	0.000006	0.002991	0.002997
5	0.000006	0.00299	0.002996
6	0.000006	0.003001	0.003007
7	0.000005	0.002995	0.003
8	0.000005	0.003001	0.003006
9	0.000005	0.003017	0.003022
10	0.000006	0.002999	0.003005
AVG	0.0000055	0.002999	0.0030045
MAX	0.000006	0.003017	0.003022
MIN	0.000005	0.00299	0.002996
MEDIAN	0.0000055	0.002997	0.003003

Polling (No delay)			
	A phase (sec)	B phase (sec)	Total (sec)
1	0.000005	0.009077	0.009082
2	0.000005	0.019026	0.019031
3	0.000005	0.009028	0.009033
4	0.000005	0.009078	0.009083
5	0.000005	0.009167	0.009172
6	0.000005	0.008978	0.008983
7	0.000005	0.009949	0.009954
8	0.000005	0.009407	0.009412
9	0.000005	0.009121	0.009126
10	0.000005	0.009057	0.009062
AVG	0.000005	0.0101888	0.010194
MAX	0.000005	0.019026	0.019031
MIN	0.000005	0.008978	0.008983
MEDIAN	0.000005	0.0090995	0.009105

Polling (1ms delay)			
	A phase (sec)	B phase (sec)	Total (sec)
1	0.000005	0.009063	0.009068
2	0.000005	0.006763	0.006768
3	0.000005	0.006755	0.00676
4	0.000004	0.008961	0.008965
5	0.000005	0.010886	0.010891
6	0.000004	0.009024	0.009028
7	0.000004	0.016488	0.016492
8	0.000006	0.00896	0.008966
9	0.000004	0.006765	0.006769
10	0.000005	0.008947	0.008952
AVG	0.0000047	0.0092612	0.0092659
MAX	0.000006	0.016488	0.016492
MIN	0.000004	0.006755	0.00676
MEDIAN	0.000005	0.0089605	0.0089655

No delay	1ms delay	No delay	interrupt
90.8974082285311%		29.4737977986619%	
1ms delay	interrupt		
32.4253445428938%			

2. Xilinx Ubuntu

Environment: ZCU106 / Ubuntu 20.04

8/11 GUI가 가능한 Petalinux가 필요하여 판교에서 배운 Petalinux를 설치하였고 이는 Ubuntu 16 기반의 OS였다. 이에 opencv-python설치가 상당히 오래 걸려 진행에 차질이 빚어졌고, 이를 개선하기 위해 Xilinx에서 제공하는 최신의 Ubuntu를 설치하여 진행해보았다. 결과 opencv-python 및 기타의 작업들이 훨씬 빠르게 설치되는 것을 확인하였고, 추가로 Vivado bitstream, module & device driver 컴파일 과정까지 확인해보았다.

A. Customize board hardware configuration

기존 SD카드 boot sector partition에 세개의 부트이미지, 이미지 파일을 변경하는 것으로는 불가능해보인다. 새로운 방법이 필요한데, target board OS에서 xlnx-config를 사용하여 가능하다.

먼저, 기존 Petalinux처럼 빌드를 완료한다. 여기서 필요한 파일들은 아래와 같다.

Filename	Description
fsbl.elf	Zynq Ultrascale+ First Stage Boot Loader
bl31.elf	ARM Trusted Firmware (ATF)
pmufw.elf	Platform Management Unit Firmware
system.bit	The Programmable Logic bitstream
system.dtb	The Linux Device tree which matches the contents bitstream
• fsbl.elf 는 zynqmp_fsbl.elf라는 파일을 이름만 바꿔서 사용함.	

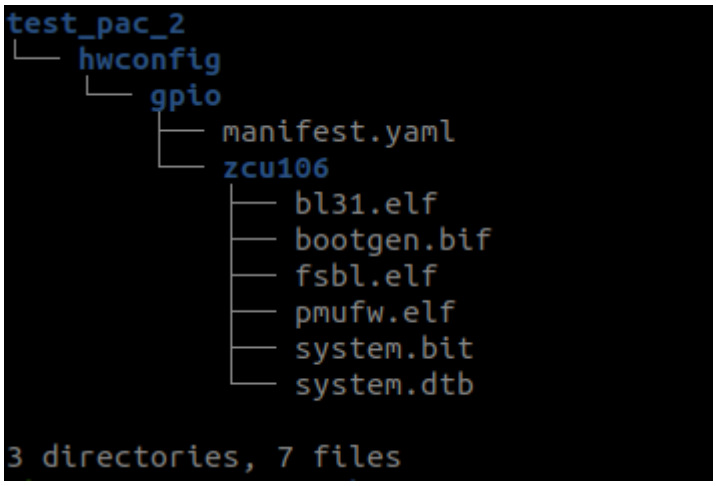
이 파일들을 모두 target board로 옮겨준다. 이후, snap을 이용하여 xlnx-config를 설치한다. 명령어는 아래와 같다.

sudo apt update

sudo apt install snapd

sudo snap install xlnx-config --classic --channel=1.x or 2.x

이후 /boot/firmware/에 xlnx-config라는 폴더를 만들어준다. 이 폴더에 다음과 같은 디렉토리를 만들어준다.



이 폴더 중 gpio, test_pac_2는 사용자 정의이다.

이 파일 중 설명이 없는 파일에 대한 설명이다.

- Manifest.yaml : 이 boot configure에 대한 설명 파일이다. 형식은 다음과 같다.

```
name: test_platform_gpio
description: Boot assets for the 2022.8 gpio design
revision: 1
assets:
        zcu106: zcu106
```

- Bootgen.bif : Bootgen config file used by xlnx-config to package new boot.bin

```
the_ROM_image:
{
        [bootloader, destination_cpu=a53-0] fsbl.elf
        [pmufw_image] pmufw.elf
        [destination_device=pl] system.bit
        [destination_cpu=a53-0, exception_level=el-3, trustzone] bl31.elf
        [destination_cpu=a53-0, load=0x00100000] system.dtb
        [destination_cpu=a53-0, exception_level=el-2] /usr/lib/u-boot/xilinx_zynqmp_virt/u-boot.elf
}
```

-> 여기서 u-boot.elf는 ubuntu 20.04의 ubuntu, 즉 해당 경로의 elf파일을 사용하여야 한다.

이렇게 디렉토리를 구성하고 xlnx-config –q 명령어를 입력하였을 때 다음과 같은 화면이 나타나면 성공적으로 등록이 된것이다.

이후 xlnx-config –a test_platform_gpio를 입력하고 특별한 error 없이 나온다면 정상적으로 적용이 된것이며, 이후 재부팅을 한번 해야 적용이 된다.

이후 devmem2를 사용하여 LED driver를 등록시키고 간단하게 led를 켜보았더니 정상적으로 켜짐을 보아 bitstream이 정상적으로 적용되었음을 확인 할 수 있었다.

추가로, 이 os의 경우 기존 petalinux와 달리 lib/modules/kernel에 device driver / module을 컴파일 할 수 있는 커널 헤더가 존재하여, 이전처럼 host에서 petalinux tool을 사용하여 module을 cross-compile 할 필요 없이 target에서 바로 compile이 가능하다.