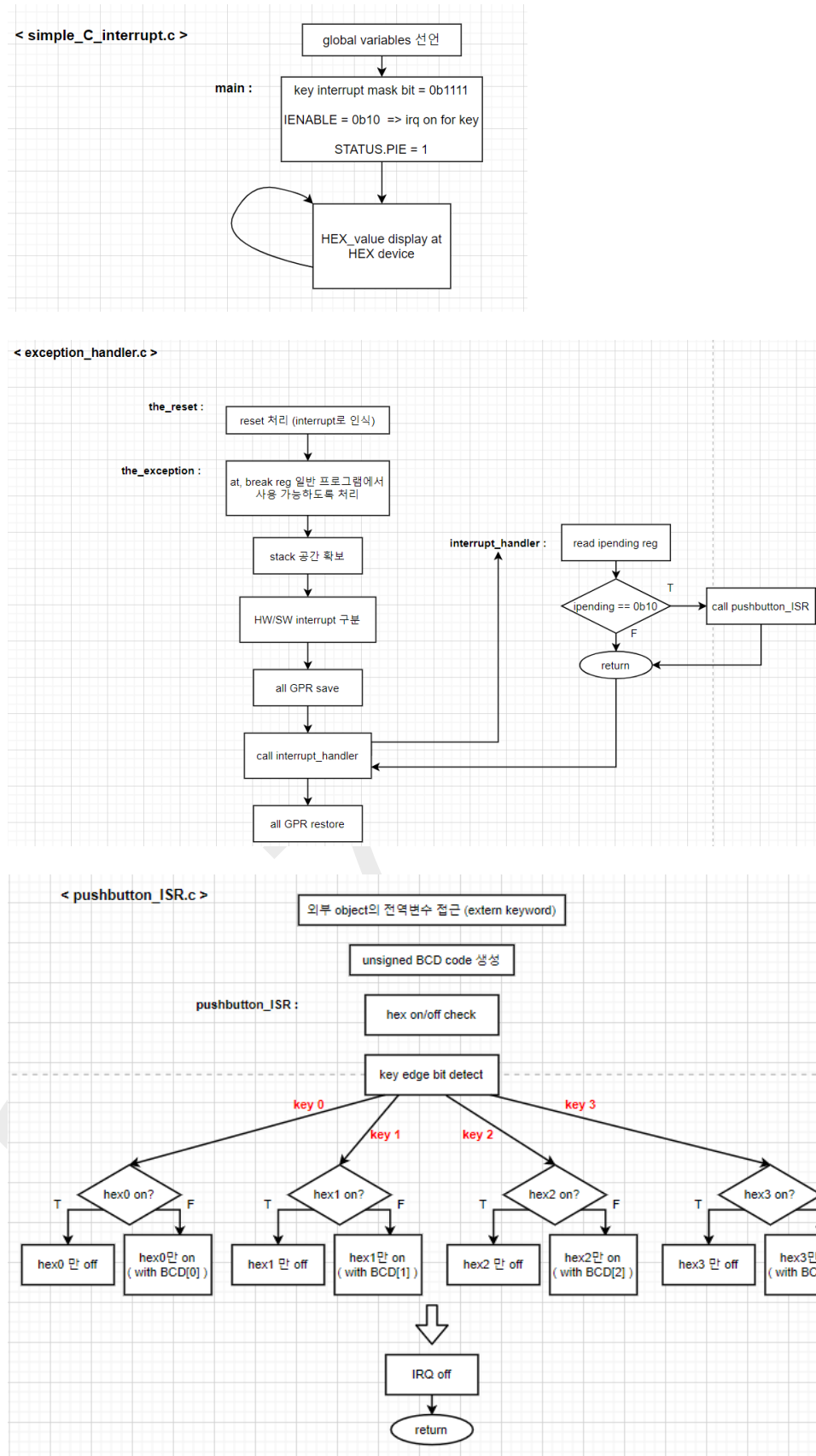


전자 HW 설계 - 실습 보고서

이름: 박관영 (2016124099)

✓ Part I

동작 원리



전자 HW 설계 – 실습 보고서

>> Lab5의 part1에서 구현한 프로그램과 동일한 동작을 구현하는 프로그램이다. C code에서 interrupt를 처리하기 위해 assembly언어를 사용하였고, nios2_ctrl_reg_macros.h를 통해 special purpose reg를 read하는 instruction을 처리하였다. Key 0-3을 통해 interrupt를 발생시키고, 각 key가 눌리면 key의 번호에 해당하는 hex에 0-3을 display하거나 off하는 동작이 수행된다. 즉, key를 통해 해당 hex를 toggle시키게 된다. 이 때, 특정 key가 해당 hex말고 다른 hex를 제어하게 되는 오류가 발생하지 않도록 처리하였다.

<simple_C_interrupt.c>

Nios2를 interrupt을 받아들일 수 있는 상태로 만들고, 각 device들의 parallel port나 global variables를 선언한다. main에서 사용하고자 하는 device를 interrupt 발생시킬 수 있는 상태로 만들고, nios2를 interrupt를 받아들일 수 있는 상태로 만든다. 즉, pushbutton interrupt mask bit를 on 하고 ienable, status.pie를 on하는 과정이다. 그 이후 IDLE상태를 반복하며 interrupt를 통해 처리된 값을 원하는 device에 load한다.

<exception_handler.c>

C 언어에서도 exception을 처리할 때에는 register 단위의 세밀한 작업이 필요하기 때문에 assembly어를 통해 처리하게 된다. 이 파일에서는 reset을 interrupt로서 처리하는 과정과 sp에 모든 reg를 save하고 해당 interrupt를 처리하는 함수를 call 한 뒤 다시 모든 reg를 restore하는 과정이 진행된다. lpending을 읽음으로써 처리하고자 하는 interrupt에 관련된 함수를 call할 수 있게 된다.

<pushbutton_ISR.c>

Exception_handler 파일을 통해 call되는 파일이다. main에서 사용했던 전역변수들에 접근하여 이를 update하는 방식으로 구현하였고, hex가 on/off상태인지를 파악한 뒤 interrupt가 발생하게 된 pushbutton의 number를 확인한다. Interrupt를 발생시킨 key에 대한 hex만 toggle한 뒤 IRQ를 turn off하고 return하게 된다.

구현 코드 설명

```
<<<<<< simple_C_interrupt.c>>>>>>
```

```
#include "address_map_nios2.h"
```

```
#include "nios2_ctrl_reg_macros.h" // 제공된 header file include
```

```
volatile int* KEY_ptr = (int*)KEY_BASE;
```

```
volatile int* HEX_ptr = (int*)HEX3_HEX0_BASE;
```

```
/* 사용할 device volatile 형태로 선언, 다른 object에서도 접근할 것이므로, static keyword는 사용하지 않았다. */
```

```
int HEX_value=0; // 이 전역변수를 update하여 hex device에 display
```

```
int main() {
```

```
    *(KEY_ptr + 2) = 0b1111; // all key interrupt mask bit on
```

```
    NIOS2_WRITE_IENABLE(0x2); // ienable level 1 bit on
```

```
    NIOS2_WRITE_STATUS(1); // status.pie bit on
```

전자 HW 설계 – 실습 보고서

```
while(1) *HEX_ptr=HEX_value;      // IDLE, display hex_value on HEX device
}

<<<<<< exception_handler.c>>>>>>

#include"nios2_ctrl_reg_macros.h"

void main(main);
void interrupt_handler(void);
void pushbutton_ISR(void);

void the_reset(void) __attribute__((section(".reset")));      // reset도 interrupt로서 처리한다
void the_reset(void) {
    asm(".set noat");      // at를 일반 프로그램에서 사용
    asm(".set nobreak");      // break reg를 일반 프로그램에서 사용
    asm("movia r2, main");
    asm("jmp r2");
}

void the_exception(void) __attribute__((section(".exceptions")));
void the_exception(void) {
    asm(".set noat");
    asm(".set nobreak");
    asm("subi sp,sp,128");      // sp space 확보
    asm("stw et, 96(sp)");
    asm("rdctl et, ctl4");      // ipending reg read
    asm("beq et, r0,SKIP_EA_DEC");      // HW/SW interrupt detect
    asm("subi ea, ea, 4");
    asm("SKIP_EA_DEC:");
    asm("stw r1, 4(sp)");      // all reg save
    asm("stw r2, 8(sp)");
    .....
    asm("stw r31, 124(sp)");
    asm("addi fp, sp, 128");
    asm("call interrupt_handler");      // call interrupt_handler
    asm("ldw r1, 4(sp)");
    asm("ldw r2, 8(sp)");
    .....
    asm("ldw r31, 124(sp)");
    asm("addi sp, sp, 128");      // all reg restore
    asm("eret");
}

void interrupt_handler(void) {
    int ipending;
```

전자 HW 설계 – 실습 보고서

```
NIOS2_READ_IPENDING(ipending);    // read ipending
if (ipending & 0x2)                 // interrupt가 key에 의해 발생했는지 검사
    pushbutton_ISR();
return;
```

<<<<<< pushbutton_ISR.c >>>>>>>

```
#include"address_map_nios2.h"
```

```
extern volatile int* KEY_ptr;        // extern keyword 사용하여 main의 KEY_ptr 에 접근
extern int HEX_value;               // extern keyword 사용하여 main의 HEX_value 에 접근
```

```
unsigned int BCD[10] = { 0b00111111, 0b00000110, 0b01011011, 0b01001111,
                        0b01100110, 0b01101101, 0b01111101, 0b00000111,
                        0b01111111, 0b01100111 };
```

/* HEX에 직접적으로 display될 BCD code */

```
void pushbutton_ISR(void) {
    int hex0, hex1, hex2, hex3;
    hex0 = (HEX_value) & 0x000000ff;
    hex1 = (HEX_value) & 0x0000ff00;
    hex2 = (HEX_value) & 0x00ff0000;
    hex3 = (HEX_value) & 0xff000000;    // to check hex on/off

    if ((*KEY_ptr + 3) & 0b0001)!=0) {    // check which key pressed
        if (hex0==0) {
            HEX_value = (HEX_value) | BCD[0];    // hex on with bcd[0], | 연산을 사용하여 해당 hex만 on
        }
        else {
            HEX_value = (HEX_value) & 0xfffff00;    // & 연산을 통해 해당 hex만 off
        }
    }
    else if ((*KEY_ptr + 3) & 0b0010)!=0) {
        if (hex1==0) {
            HEX_value = (HEX_value) | BCD[1]<<8; // hex on with bcd[1], | 연산을 사용하여 해당 hex만 on
        }
        else {
            HEX_value = (HEX_value) & 0xffff00ff;    // & 연산을 통해 해당 hex만 off
        }
    }
    else if ((*KEY_ptr + 3) & 0b0100)!=0) {
        if (hex2==0) {
            HEX_value = (HEX_value) | BCD[2]<<16; // hex on with bcd[2], | 연산을 사용하여 해당 hex만 on
        }
    }
}
```

전자 HW 설계 – 실습 보고서

```
        else {
            HEX_value = (HEX_value) & 0xff00ffff; ;    // & 연산을 통해 해당 hex만 off
        }
    }
    else if ((*KEY_ptr + 3) & 0b1000)!=0) {
        if (hex3==0) {
            HEX_value = (HEX_value) | BCD[3]<<24;    // hex on with bcd[2], | 연산을 사용하여 해당 hex만 on
        }
        else {
            HEX_value = (HEX_value) & 0x00ffffff;    // & 연산을 통해 해당 hex만 off
        }
    }

    int press = *(KEY_ptr + 3);
    *(KEY_ptr + 3) = press;        // IRQ off

    return;
}
```

결과 및 토의

```
while(1) *HEX_ptr=HEX_value;
```

```
ldw    r3, -32736(gp)
```

```
ldw    r2, -32752(gp)
```

```
stwio  r3, 0(r2)
```

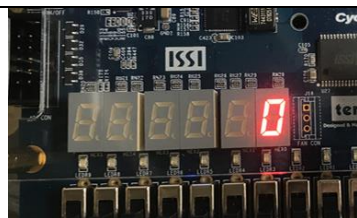
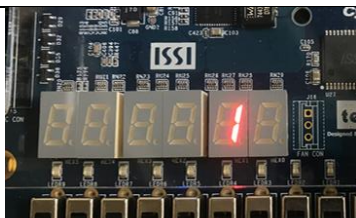

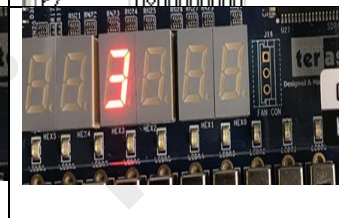
```
br     -0x10 (0x000003D8)
```

>>> continue하는 동안 아무 interrupt이 발생하지 않는다면, IDLE상태를 계속 머무는 것을 볼 수 있다. HEX_value를 HEX device에 write하고 있다.

...	...	ra	0x000004B4
status	0x00000000	status	0x00000000
estatus	0x00000001	estatus	0x00000001
bstatus	0xFFFFFFFF	bstatus	0xFFFFFFFF
ienable	0x00000002	ienable	0x00000002
ipending	0x00000000	ipending	0x00000002
cpuid	0x00000000	cpuid	0x00000000

Key를 눌러 interrupt이 걸리게 되면, ipending 값이 2로 바뀌게 되는 것을 볼 수 있다.

전자 HW 설계 - 실습 보고서

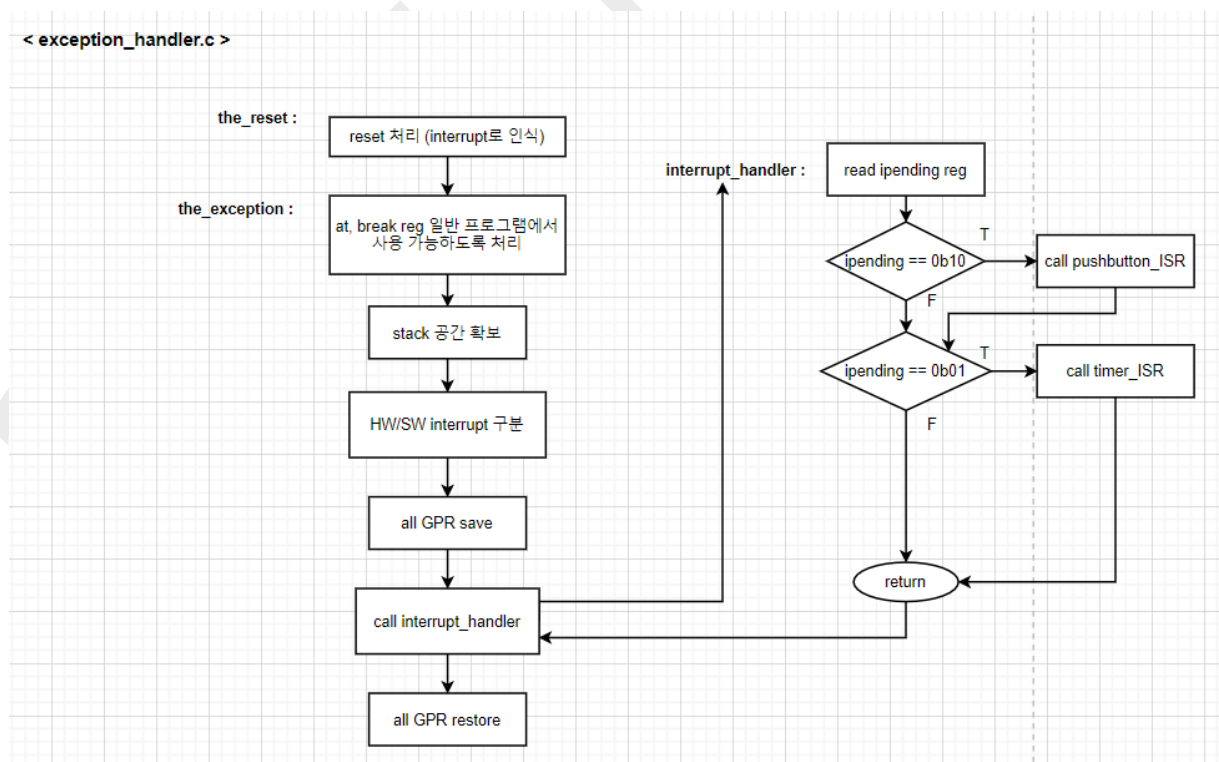
<div><div>Registers</div><div><div>Reg</div><div>Value</div></div><div><div>pc</div><div>0x000003DC</div></div><div><div>zero</div><div>0x00000000</div></div><div><div>r1</div><div>0x00000000</div></div><div><div>r2</div><div>0xFF200020</div></div><div><div>r3</div><div>0x0000003F</div></div><div><div>r4</div><div>0x00000000</div></div><div><div>r5</div><div>0x00000000</div></div></div>	<div><div>Registers</div><div><div>Reg</div><div>Value</div></div><div><div>pc</div><div>0x000003E0</div></div><div><div>zero</div><div>0x00000000</div></div><div><div>r1</div><div>0x00000000</div></div><div><div>r2</div><div>0xFF200020</div></div><div><div>r3</div><div>0x00000060</div></div><div><div>r4</div><div>0x00000000</div></div><div><div>r5</div><div>0x00000000</div></div></div>	<div><div>Registers</div><div><div>Reg</div><div>Value</div></div><div><div>pc</div><div>0x000003E0</div></div><div><div>zero</div><div>0x00000000</div></div><div><div>r1</div><div>0x00000000</div></div><div><div>r2</div><div>0xFF200020</div></div><div><div>r3</div><div>0x00580000</div></div><div><div>r4</div><div>0x00000000</div></div><div><div>r5</div><div>0x00000000</div></div><div><div>r6</div><div>0x00000000</div></div></div>	<div><div>Registers</div><div><div>Reg</div><div>Value</div></div><div><div>pc</div><div>0x000003DC</div></div><div><div>zero</div><div>0x00000000</div></div><div><div>r1</div><div>0x00000000</div></div><div><div>r2</div><div>0xFF200020</div></div><div><div>r3</div><div>0x4F000000</div></div><div><div>r4</div><div>0x00000000</div></div><div><div>r5</div><div>0x00000000</div></div><div><div>r6</div><div>0x00000000</div></div></div>
			
R3를 통해서 bcd[0]의 값이 hex_value에 저장된 것을 볼 수 있다.	R3를 통해서 bcd[1]<<8의 값이 hex_value에 저장된 것을 볼 수 있다.	R3를 통해서 bcd[2]<<16의 값이 hex_value에 저장된 것을 볼 수 있다.	R3를 통해서 bcd[3]<<24의 값이 hex_value에 저장된 것을 볼 수 있다.

<https://youtu.be/Mw3k-B-mA5M>

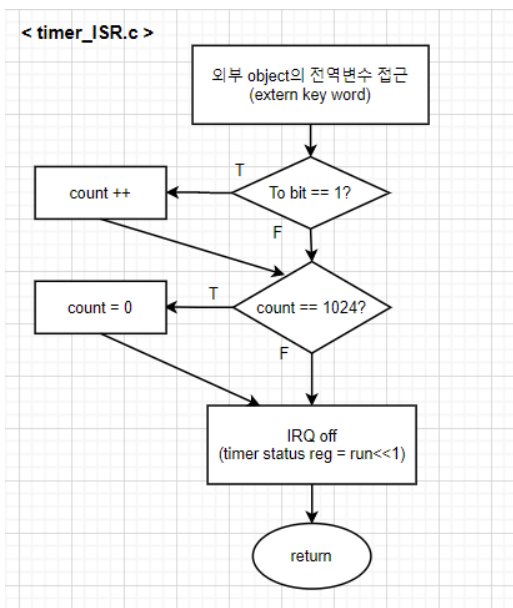
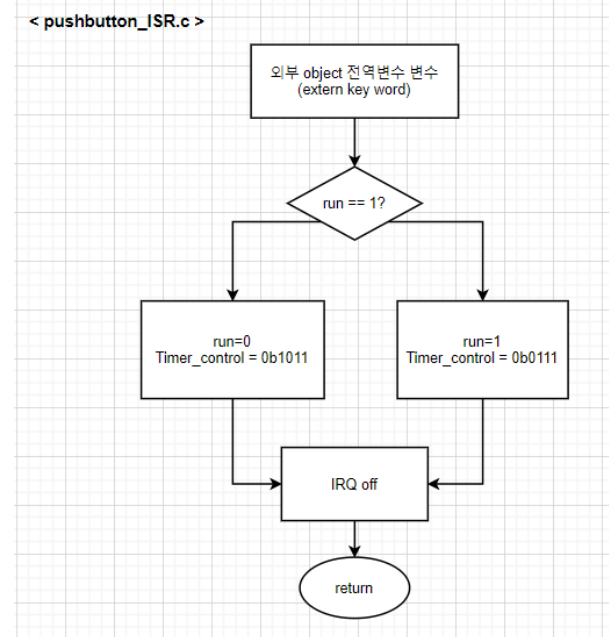
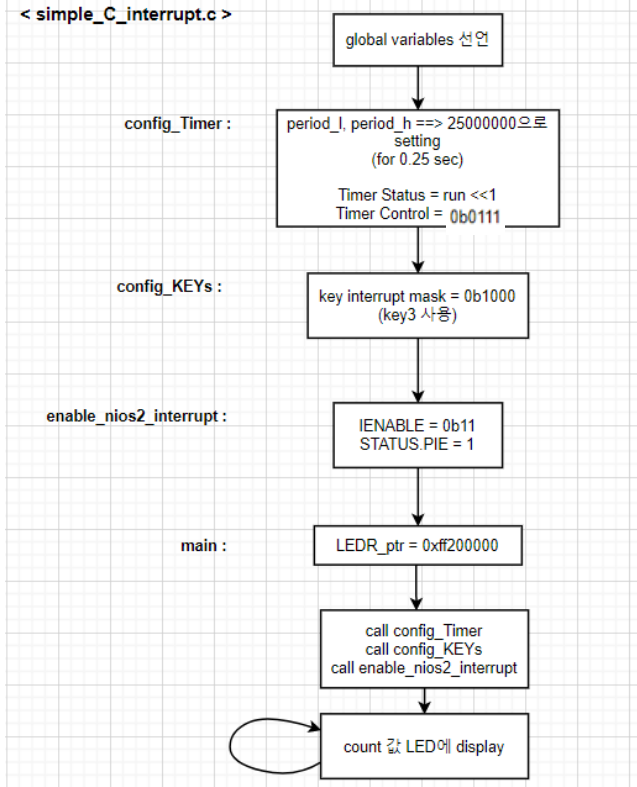
>>> 실험 결과 동영상 첨부합니다.

✓ Part II

동작 원리



전자 HW 설계 – 실습 보고서



Lab5 part2와 같은 timer를 동작하는 프로그램이다. 단, 이 프로그램에서는 timer 초가 올라가는 시간을 0.25sec로 설정하였다. Part1과 마찬가지로 interrupt 처리를 위해 assembly어를 사용하였고, 전역변수를 통해 device에 입력할 값을 제어하였다. Interval timer와 key를 통해 interrupt를 발생시킬 수 있고, 각 device에 의해 발생한 interrupt 처리를 위한 함수를 각각 정의하였다.

<simple_C_interrupt.c>

Part1과 마찬가지로 IDLE상태를 머물게 되고, 사용하고자 하는 device의 memory mapped register의 초기값을 설정해주는 단계이다. 이 때, key에 대해서는 interrupt mask bit를 on 시키는데, key3에 대한 interrupt만 가능하도록 0b1000의 값으로 초기화 한다. Interval timer에 대해서는 period_l,h에 25000000의 값을 halfword 단위로 입력하여 0.25sec 단위로 timer의 시간이 올라가도록 하였고, status 는 전역변수 run bit을 그대로 읽어서 초기화 하였다. Control reg는 0b0111의 값으로 초

전자 HW 설계 – 실습 보고서

기화 하여 interrupt를 발생시키고, continue상태를 유지하며 start상태로 초기화 할 수 있었다. main에서는 lenable에 0b11의 값을 넣어 nios2가 timer, key에 대한 interrupt를 받을 수 있는 상태로 하고 status.pie bit를 on 상태로 한다. IDLE 상태를 머물면서 LEDR data reg에 count 전역변수의 값을 계속 write하게 된다.

<exception_handler.c>

Part1과 동일한 형태이지만, interrupt_handler 함수에서 ipending을 검사할 때, timer bit인지 key bit인지를 검사하여 그에 맞는 function을 call하여 interrupt를 처리하게 된다.

<timer_ISR.c>

Timer의 status reg의 To bit를 검사하여 1이라면, count++을 통해 timer sec를 증가하고 만약 LEDR에 표현할 수 있는 수 1024까지 세었다면 count를 0으로 초기화 한다. 그리고 status reg에 run bit를 다시 write하여 timer IRQ를 off하게 된다.

<pushbutton_ISR.c>

전역변수 run 값을 검사하여 그 값을 toggle시키고, run=1이라면 그 값을 toggle시킨 뒤 timer를 stop한다. Run=0이라면 그 값을 toggle시킨 뒤 timer를 start한다. 이를 통해 key interrupt로 timer의 start/stop을 toggle시킬 수 있다.

구현 코드 설명

```
<<<<< simple_C_interrupt.c >>>>>

#include "address_map_nios2.h"
#include "nios2_ctrl_reg_macros.h"

volatile int* KEY_ptr = (int*)KEY_BASE;
volatile int* Timer_ptr = (int*)TIMER_BASE;
/* 사용할 device의 주소를 volatile int type으로 선언, 다른 object에서 접근할 수 있다. */

int count=0;
int run=1;           // device에 write할 전역변수 선언.

void config_timer(void){
    *(Timer_ptr+2)=0x7840;
    *(Timer_ptr+3)=0x17d;
    *Timer_ptr=(run<1);
    *(Timer_ptr+1)=0b0111;           // timer register initialization
}

void config_KEYS(void){
    *(KEY_ptr + 2) = 0b01000;           // key3 interrupt mask bit on
}

void enable_nios2_interrups(void){
```


전자 HW 설계 – 실습 보고서

```
NIOS2_WRITE_IENABLE(0x3);      // key, timer에 대한 interrupt 받을 수 있게 한다.
NIOS2_WRITE_STATUS(1);         // status.pie on
}

int main() {
    volatile int* LEDR_ptr=(int*)0xff200000;      // LEDR device volatile int type으로 선언
    config_timer();
    config_KEYS();
    enable_nios2_interrupts();
    while(1) *LEDR_ptr=count;                    // IDLE상태에서 count값 LEDR에 write
}
```

<<<<< exception_handler.c >>>>>

```
#include"nios2_ctrl_reg_macros.h"
```

```
void main(main);
void interrupt_handler(void);
void pushbutton_ISR(void);
void timer_ISR(void);
```

```
void the_reset(void) __attribute__((section(".reset")));
```

```
void the_reset(void) {
```

```
    asm(".set noat");
    asm(".set nobreak");
    asm("movia r2, main");
    asm("jmp r2");
}
```

```
void the_exception(void) __attribute__((section(".exceptions")));
```

```
void the_exception(void) {
```

```
    asm(".set noat");
    asm(".set nobreak");
    asm("subi sp,sp,128");
    asm("stw et, 96(sp)");
    asm("rdctl et, ctl4");
    asm("beq et, r0,SKIP_EA_DEC");
    asm("subi ea, ea, 4");
    asm("SKIP_EA_DEC:");
    asm("stw r1, 4(sp)");
    asm("stw r2, 8(sp)");
```

.....

```
    asm("stw r31, 124(sp)");
    asm("addi fp, sp, 128");
    asm("call interrupt_handler");
```

전자 HW 설계 – 실습 보고서

```
asm("ldw r1, 4(sp)");
asm("ldw r2, 8(sp)");

.....
asm("ldw r31, 124(sp)");
asm("addi sp, sp, 128");
asm("eret");
}

void interrupt_handler(void) {
    int ipending;
    NIOS2_READ_IPENDING(ipending);
    if (ipending & 0x2)                // pushbutton ipending bit 검사
        pushbutton_ISR();
    else if (ipending & 0x1)           // timer ipending bit 검사
        timer_ISR();
    return;
}

<<<<<< pushbutton_ISR.c >>>>>>

#include"address_map_nios2.h"

extern volatile int* KEY_ptr;
extern volatile int* Timer_ptr;
extern int count;
extern int run;                      // extern keyword를 통해 main object의 전역변수 접근

void pushbutton_ISR(void) {

    if(run){
        run=0;
        *(Timer_ptr) = (run << 1);    // run bit toggle 후 timer status update
    }
    else{
        run=1;
        *(Timer_ptr) = (run << 1);    // run bit toggle 후 timer status update
    }
    int press = *(KEY_ptr+3);
    *(KEY_ptr+3) = press;

    return;
}

<<<<<< timer_ISR.c >>>>>>
```

전자 HW 설계 – 실습 보고서

```
#include"address_map_nios2.h"

extern volatile int* KEY_ptr;
extern volatile int* Timer_ptr;
extern int count;
extern int run;

void timer_ISR(void) {
    if (run) { // run 값이 1일때만 동작( stop 상태일 때는 동작을 하지 않는다. )
        count++; // To bit가 1이면 count++
        if (count == 0x400) count = 0; // LEDR 범위를 벗어나면, count 초기화
    }
    *(Timer_ptr) = (run << 1); // run bit 그대로 timer_status reg에 write
    return;
}
```

결과 및 토의

```
while(1) *LEDR_ptr=count;
orhi r3, zero, 0xFF20
ldw r2, -32732(gp)
stwio r2, 0(r3)
br -0xC (0x00000340)
```

>>> IDLE 상태에 계속 머물면서 count 값을 LEDR device에 write하는 것을 확인할 수 있었다.

Reg	Value
pc	0x00000348
zero	0x00000000
r1	0x00000000
r2	0x00000066
r3	0xFF200000
r4	0x00000000
r5	0x00000000
r6	0x00000000

Reg	Value
pc	0x00000344
zero	0x00000000
r1	0x00000000
r2	0x00000078
r3	0xFF200000
r4	0x00000000
r5	0x00000000
r6	0x00000000
r7	0x00000000

>>> r2의 값이 LEDR에 표현되는 이진수의 값과 동일한 것을 통해 count의 값이 r2를 통해 LEDR에 write된다는 것을 확인할 수 있었다. 또한 r3에 LEDR device의 pointer가 저장되는 것을 확인하였다.

```
ea 0x00000344
ba 0xFFFFFFFF
ra 0x0000033C
status 0x00000001
estatus 0x00000001
bstatus 0xFFFFFFFF
ienable 0x00000003
ipending 0x00000000
cpuid 0x00000000
```

>>> ienable값이 0x3으로 저장되어있는 것을 볼 수 있다.

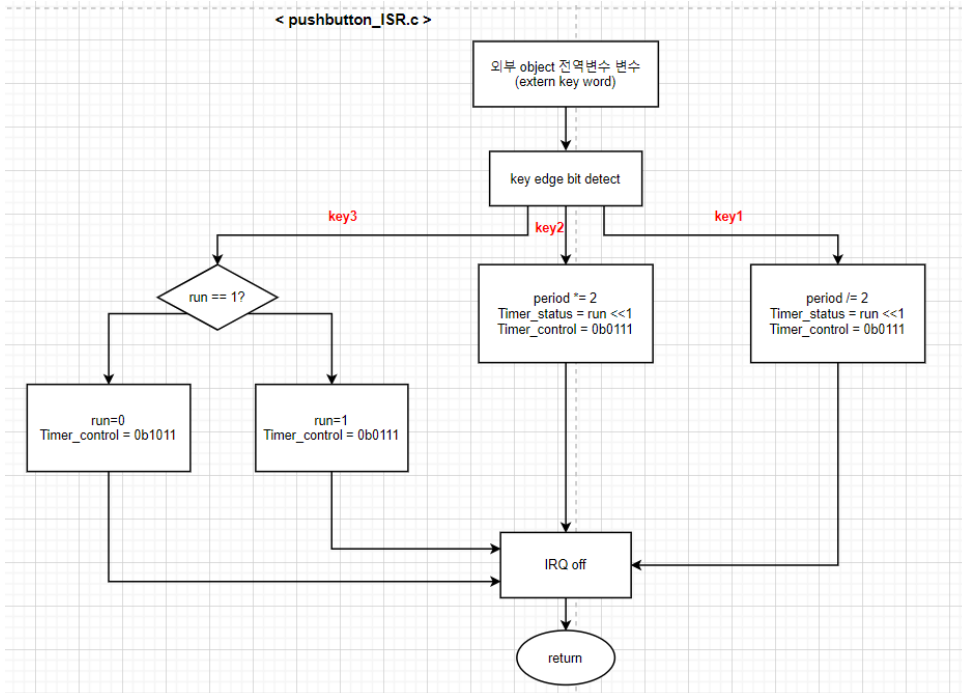
전자 HW 설계 – 실습 보고서

<https://youtu.be/AR0I7KTa7D4>

>>> 실험 결과 동영상 첨부합니다.

✓ Part III

동작 원리



Part2와 마찬가지로 timer를 구현하는 프로그램이다. 그러나 key interrupt를 처리하는 과정에서 몇 가지 기능을 추가하였다. Key1, 2를 통해 timer의 속도를 조절하는 기능이다. Key1을 통해 timer의 초 증가 속도를 두 배 빠르게 하고, key2를 통해 두 배 느리게 하도록 구현하였다. 이 때 사용되는 방법은 period를 반으로 감소시켜 속도를 두 배 빠르게 할 수 있었고, 반대의 방법으로 속도를 두 배 느리게 할 수 있었다. Period를 update한 이후에는 다시 interval timer memory mapped register를 시간이 흘러가도록 초기화 하였다.

구현 코드 설명

```
<<<<< pushbutton_ISR.c >>>>>>
```

```
#include"address_map_nios2.h"
```

```
extern volatile int* KEY_ptr;
```

```
extern volatile int* Timer_ptr;
```

```
extern int count;
```

```
extern int run; // main의 전역변수 접근
```

```
void pushbutton_ISR(void) {
```

```
    int Period=0; // period 변수 초기화
```

전자 HW 설계 - 실습 보고서

```
if(*(KEY_ptr+3)==0b1000){
    if(run){
        run=0;
        *(Timer_ptr+1)=0b1011;
    }
    else{
        run=1;
        *(Timer_ptr+1)=0b0111;
    }
    // part1과 동일한 code
}
else if(*(KEY_ptr+3)==0b0010){ // key1 pressed - faster
    Period=*(Timer_ptr+2);
    Period=Period|(*(Timer_ptr+3)<16); //halfword의 period reg에서 period 값을 읽고 하나의 값으로 합침.
    Period=Period/2; // period값 절반으로 나눈다.
    *(Timer_ptr+2)=Period;
    *(Timer_ptr+3)=Period>>16; // period 값 update
    *Timer_ptr=(run<1);
    *(Timer_ptr+1)=0b0111; // interval timer reg initialization
}
else if(*(KEY_ptr+3)==0b0100){ // key2 pressed - slow
    Period=*(Timer_ptr+2);
    Period=Period|(*(Timer_ptr+3)<16); //halfword의 period reg에서 period 값을 읽고 하나의 값으로 합침.
    Period=Period*2; // period값 두 배로 증가.
    *(Timer_ptr+2)=Period;
    *(Timer_ptr+3)=Period>>16; // period값 update
    *Timer_ptr=(run<1);
    *(Timer_ptr+1)=0b0111; // interval timer reg initialization
}

int press = *(KEY_ptr+3);
*(KEY_ptr+3) = press; // IRQ off

return;
}
```

결과 및 토의

```
while(1) *LEDR_ptr=count;
orhi r3, zero, 0xFF20
ldw r2, -32736(gp)
stwio r2, 0(r3)
br -0xC (0x00000414)
```

>>> IDLE 상태에 머물면서 count값을 LEDR에 write하는 것을 확인할 수 있다.

전자 HW 설계 – 실습 보고서

Registers		Registers	
Reg	Value	Reg	Value
pc	0x0000041C	pc	0x00000418
zero	0x00000000	zero	0x00000000
r1	0x00000000	r1	0x00000000
r2	0x000001C0	r2	0x00000108
r3	0xFF200000	r3	0xFF200000
r4	0x00000000	r4	0x00000000
r5	0x00000000	r5	0x00000000
r6	0x00000000	r6	0x00000000
r7	0x00000000	r7	0x00000000

>>> r2가 시간에 따라 증가하는 것을 통해 r2로 count값을 읽어 LEDR에 write하는 것을 확인할 수 있었다. 또한, r3를 통해 LEDR의 주소를 저장하는 것을 확인할 수 있었다.

ra	0x00000410
status	0x00000001
estatus	0x00000001
bstatus	0xFFFFFFFF
ienable	0x00000003
ipending	0x00000000
cpuid	0x00000000

>>> part2와 마찬가지로 ienable에 0x3이 저장되어 key와 timer에 대해서 interrupt이 허용되어 있는 것을 확인할 수 있다.

<https://youtu.be/3fRzfDu-ad8>

>>> 실험 결과 영상 첨부합니다.