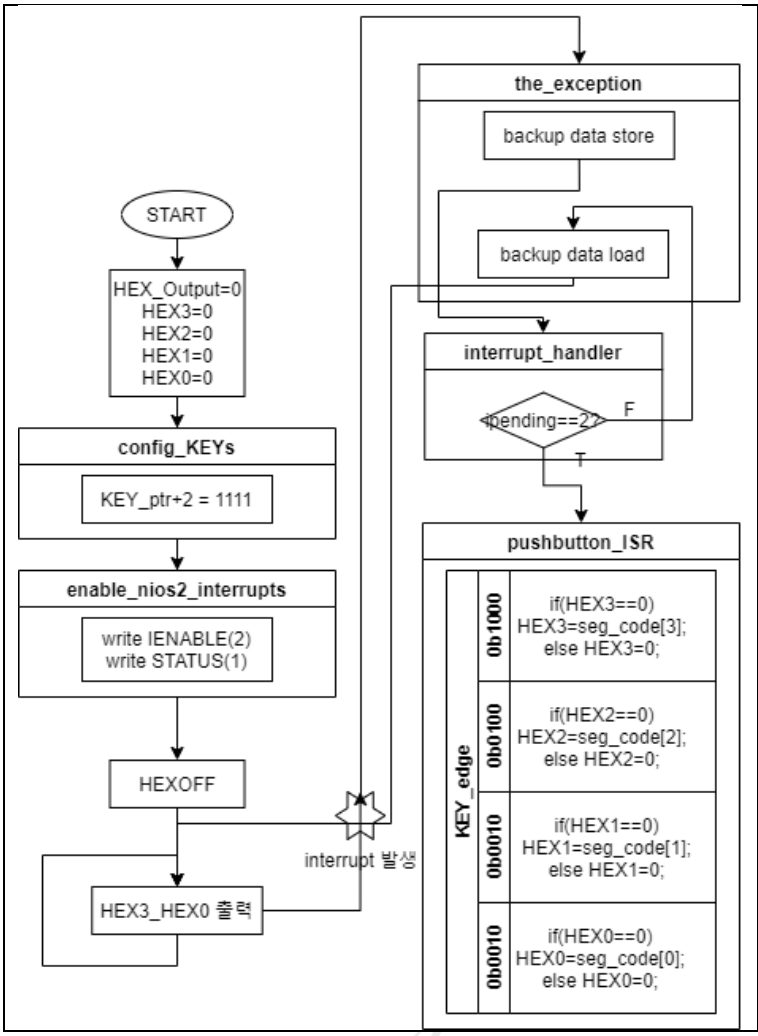
전자 HW 설계 – 실습 보고서

이름: 양해찬 (2016124145)

✓ Part I

동작 원리



Lab7은 Lab6에서 사용한 Polled IO 방법이 아닌 Interrupt IO를 사용한다.
Part1은 key3, key2, key1, key0을 누를 때 마다 각각 HEX3에 3, HEX2에 2, HEX1에 1, HEX0에 0이 토글되어 출력되도록 한다.
Main code에서 config_KEYs와 enable_nios2_interrups를 통해 key를 눌렀을 때 interrupt가 발생해 supervisor모드에서 control 할 수 있게 설정한다. while문을 통해 지속적으로 HEX 출력을 update할 수 있게 한다.

Exeption_handler code에서는 백업 데이터들(레지스터 값)을 모두 store한 후 ipending이 2이면(pushbutton에 의한 interrupt 의미)pushbutton_ISR을 call하고 pushbutton_ISR이 return 되면 다시 백업 데이터들을 load한다.

Pushbutton_ISR code에서는 KEY의 edgecapture 값으로 어떤 KEY가 눌렸는지 확인하고 이에 맞는 segment code를 각각 HEX3~HEX0에 할당한다.

구현 코드 설명

```
        //Lab7_Part1_main.c
#include <stdio.h>
#include "nios2_ctrl_reg_macros.h"
#include "address_map_nios2.h"

volatile int *KEY_ptr = (int*)KEY_BASE;
volatile int *HEX3_HEX0_ptr = (int*)HEX3_HEX0_BASE;//각각의 BASSE에 맞는 포인터
unsigned int HEX_Output=0;//HEX 초기화, 각 HEX출력값 저장하기 위한 전역변수
unsigned int HEX3=0,HEX2=0,HEX1=0,HEX0=0;//각 HEX출력값 초기화
void enable_nios2_interrupts(void); //interrupt를 enable하게 할 함수 선언
void config_KEYs(void);// KEY초기 설정 함수
void main(void){
```

```
    config_KEYs(); // configure pushbutton KEYs to generate interrupts
    enable_nios2_interrupts(); // enable interrupts in the Nios II processor
    *HEX3_HEX0_ptr=0;            //HEX OFF
    while (1){
        *HEX3_HEX0_ptr=HEX_Output;
    } // wait for an interrupt
}
/* Set up the pushbutton KEYs port in the FPGA */
void config_KEYs(void){
    *(KEY_ptr + 2)=0b01111;     //KEY0-3사용
}
/* Enable interrupts in the Nios II processor */
void enable_nios2_interrupts(void){
    NIOS2_WRITE_IENABLE(0x02);   //IRQ pushbutton
    NIOS2_WRITE_STATUS(0b01);      //interrupt.PIE=1
}
//Lab7_Part1_pushbutton_ISR.c
#include "address_map_nios2.h"
extern volatile int *KEY_ptr; // main.c 에서 선언했던 KEY_BASE 에 대한 포인터 변수
extern int HEX_Output;        // main.c 에서 선언했던 HEX3_HEX0 에 저장될 변수
extern unsigned int HEX3,HEX2,HEX1,HEX0;
unsigned int seg_code[] = { 0b00111111, 0b00000110, 0b01011011, 0b01001111,
                            0b01100110, 0b01101101, 0b01111101, 0b00000111,
                            0b01111111, 0b01100111 };//HEX출력을 위한 segcode
void pushbutton_ISR(void){
    int KEY_edge=*(KEY_ptr+3);//KEY edgecapture
    *(KEY_ptr+3)=KEY_edge;//edgecapture초기화
    if(KEY_edge==0b1000){//key3번이 눌리면
        if(HEX3==0)//이전에 OFF된 상태라면
        HEX3=seg_code[3];//hex3에 3 on
        else
        HEX3=0;         //아니라면 off
    }
    else if(KEY_edge==0b0100){//key2번이 눌리면
        if(HEX2==0)//이전에 OFF된 상태라면
        HEX2=seg_code[2];//hex2에 2 on
        else
        HEX2=0;      //아니라면 off
    }
    else if(KEY_edge==0b0010){//key1번이 눌리면
        if(HEX1==0)//이전에 OFF된 상태라면
        HEX1=seg_code[1];//hex1에 1 on
        else
        HEX1=0;         //아니라면 off
    }
```

```
    else if(KEY_edge==0b0001){//key0번이 눌리면
        if(HEX0==0)//이전에 OFF된 상태라면
        HEX0=seg_code[0];//hex0에 0 on
        else
        HEX0=0;   //아니라면 off
    }
    HEX_Output=(HEX3<<24)|(HEX2<<16)|(HEX1<<8)|(HEX0);
    //HEX_Output에 각각의 HEX출력값들을 합쳐서 저장
    return;
}
//Lab7_Part1_exception_handler.c
#include "nios2_ctrl_reg_macros.h"
/* function prototypes */
void main(void);
void interrupt_handler(void);
void pushbutton_ISR(void);
/* The assembly language code below handles Nios II reset processing */
void the_reset (void) __attribute__ ((section (".reset")));
void the_reset (void)
/*****************************************************************************
* Reset code; by using the section attribute with the name ".reset" we allow the linker program
* to locate this code at the proper reset vector address. This code just calls the main program
*****************************************************************************/
{
asm (".set noat"); // magic, for the C compiler
asm (".set nobreak"); // magic, for the C compiler
asm ("movia r2, main"); // call the C language main program
asm ("jmp r2");
}

/* The assembly language code below handles Nios II exception processing. This code should not be
* modified; instead, the C language code in the function interrupt_handler() can be modified as
* needed for a given application. */
void the_exception (void) __attribute__ ((section (".exceptions")));
void the_exception (void)
/*****************************************************************************
* Exceptions code; by giving the code a section attribute with the name ".exceptions" we allow
* the linker to locate this code at the proper exceptions vector address. This code calls the
* interrupt handler and later returns from the exception.
*****************************************************************************/
{
asm (".set noat"); // magic, for the C compiler
asm (".set nobreak"); // magic, for the C compiler
asm ( "subi sp, sp, 128");
asm ( "stw et, 96(sp)");     //r24
```

```
asm ( "rdctl et, ctl4");    //et=ipendig
asm ( "beq et, r0, SKIP_EA_DEC"); // interrupt is not external(SW interrupt)
asm ( "subi ea, ea, 4"); /* must decrement ea by one instruction for external
* interrupts, so that the instruction will be run(HW interrupt이기 때문에 pc-4) */
asm ( "SKIP_EA_DEC:" );
asm (   "stw    r1,  4(sp)" );                  // Save all registers
asm (   "stw    r2,  8(sp)" );
asm (   "stw    r3,  12(sp)" );
asm (   "stw    r4,  16(sp)" );
asm (   "stw    r5,  20(sp)" );
asm (   "stw    r6,  24(sp)" );
asm (   "stw    r7,  28(sp)" );
asm (   "stw    r8,  32(sp)" );
asm (   "stw    r9,  36(sp)" );
asm (   "stw    r10, 40(sp)" );
asm (   "stw    r11, 44(sp)" );
asm (   "stw    r12, 48(sp)" );
asm (   "stw    r13, 52(sp)" );
asm (   "stw    r14, 56(sp)" );
asm (   "stw    r15, 60(sp)" );
asm (   "stw    r16, 64(sp)" );
asm (   "stw    r17, 68(sp)" );
asm (   "stw    r18, 72(sp)" );
asm (   "stw    r19, 76(sp)" );
asm (   "stw    r20, 80(sp)" );
asm (   "stw    r21, 84(sp)" );
asm (   "stw    r22, 88(sp)" );
asm (   "stw    r23, 92(sp)" );     // r24 = et, 앞에서 이미 store함
asm (   "stw    r25, 100(sp)" );    // r25 = bt (skip r24 = et, because it is saved above)
asm (   "stw    r26, 104(sp)" );    // r26 = gp
// skip r27 because it is sp, and there is no point in saving this
asm (   "stw    r28, 112(sp)" );    // r28 = fp
asm (   "stw    r29, 116(sp)" );    // r29 = ea
asm (   "stw    r30, 120(sp)" );    // r30 = ba
asm (   "stw    r31, 124(sp)" );    // r31 = ra
asm (   "addi   fp,  sp, 128" );
asm (   "call interrupt_handler" ); // call the C language interrupt handler

asm (   "ldw    r1,  4(sp)" );          // Restore all registers
asm (   "ldw    r2,  8(sp)" );
asm (   "ldw    r3,  12(sp)" );
asm (   "ldw    r4,  16(sp)" );
asm (   "ldw    r5,  20(sp)" );
asm (   "ldw    r6,  24(sp)" );
asm (   "ldw    r7,  28(sp)" );
```

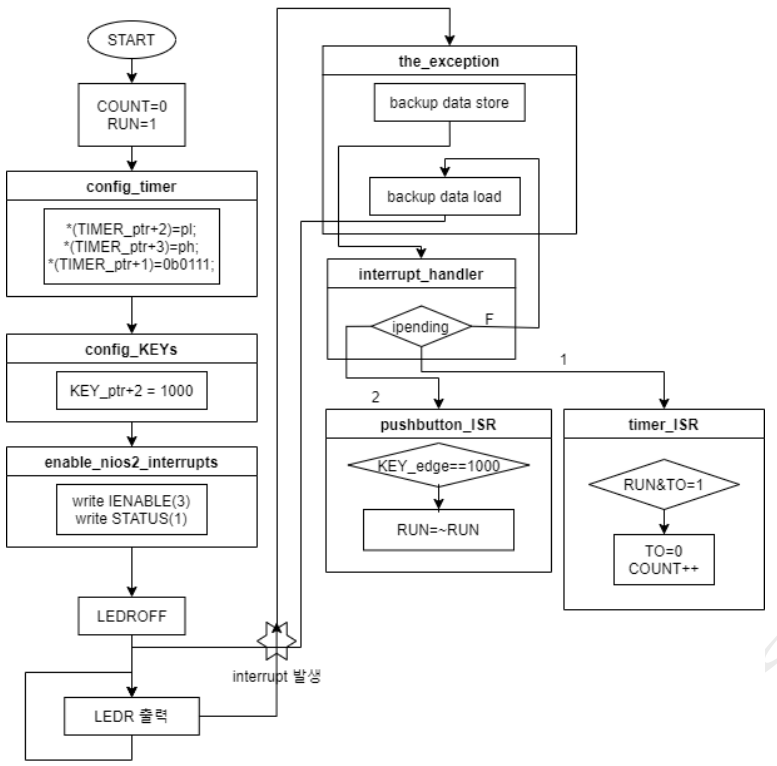
```
asm (   "ldw    r8,  32(sp)" );
asm (   "ldw    r9,  36(sp)" );
asm (   "ldw    r10, 40(sp)" );
asm (   "ldw    r11, 44(sp)" );
asm (   "ldw    r12, 48(sp)" );
asm (   "ldw    r13, 52(sp)" );
asm (   "ldw    r14, 56(sp)" );
asm (   "ldw    r15, 60(sp)" );
asm (   "ldw    r16, 64(sp)" );
asm (   "ldw    r17, 68(sp)" );
asm (   "ldw    r18, 72(sp)" );
asm (   "ldw    r19, 76(sp)" );
asm (   "ldw    r20, 80(sp)" );
asm (   "ldw    r21, 84(sp)" );
asm (   "ldw    r22, 88(sp)" );
asm (   "ldw    r23, 92(sp)" );
asm (   "ldw    r24, 96(sp)" );
asm (   "ldw    r25, 100(sp)" );    // r25 = bt
asm (   "ldw    r26, 104(sp)" );    // r26 = gp
// skip r27 because it is sp, and we did not save this on the stack
asm (   "ldw    r28, 112(sp)" );    // r28 = fp
asm (   "ldw    r29, 116(sp)" );    // r29 = ea
asm (   "ldw    r30, 120(sp)" );    // r30 = ba
asm (   "ldw    r31, 124(sp)" );    // r31 = ra
asm (   "addi   sp,  sp, 128" );
asm (   "eret" );
}
/*****************************************************************************
* Interrupt Service Routine: Determines the interrupt source and calls the appropriate subroutine
*****************************************************************************/
void interrupt_handler(void)
{
int ipending;
NIOS2_READ_IPENDING(ipending);
if ( ipending & 0x2 ) // pushbuttons are interrupt level 1
pushbutton_ISR( );//pushbutton에 의한 interrupt일 때
// else, ignore the interrupt
return;
}
```

결과 및 토의

https://www.youtube.com/watch?v=UpBt64QnpE8
KEY3, 2, 1, 0을 누를 때 마다 각각 3, 2, 1, 0이 HEX 3, 2, 2, 0으로 토글되고 있는 것을 볼 수 있다.

✓ Part II

동작 원리



Part2는 0.25초가 지날 때 마다 timer를이용해 COUNT하고 그 값을 LEDR로 출력한다. KEY 3을 누를 때 마다 RUN변수를 토글하고, 이것을 통해 COUNT를 STOP&START 할 수 있다.

Main code에서 config_KEYs와, config_timer, enable_nios2_interrups를 통해 key를 눌렀을 때와 timer가 0.25초 지날 때 마다 interrupt가 발생해 supervisor모드에서 control 할 수 있게 설정한다.

while문을 통해 지속적으로 LEDR 출력을 update할 수 있게 한다.

Exeption_handler code에서는 백업 데이터들(레지스터 값)을 모두 store한 후 ipending이 2이면 (pushbutton에 의한 interrupt 의미)pushbutton_ISR을 call하고, 1이면 timer_ISR을 call한다. Timer_ISR과 pushbutton_ISR이 return 되면 다시 백업 데이터들을 load한다.

Pushbutton_ISR code에서는 KEY3이 눌릴 때 마다 RUN이 토글되게 한다

Timer_ISR code에서는 RUN과 TO가 모두 1일 때 마다 TO를 초기화 해주고, COUNT한다.

구현 코드 설명

```
//Lab7_Part2_main.c
#include <stdio.h>
#include "nios2_ctrl_reg_macros.h"
#include "address_map_nios2.h"

int COUNT = 0; // global counter for red lights
int RUN = 1; // global, used to increment/not the count variable
volatile int *KEY_ptr = (int*)KEY_BASE;
volatile int *LEDR_ptr = (int*)LEDR_BASE;
volatile int *TIMER_ptr = (int*)TIMER_BASE;//각각의 BASSE에 맞는 포인터
void enable_nios2_interrupts(void);
void config_KEYs(void);
void config_timer(void);
void main(void){
    config_KEYs(); // configure pushbutton KEYs to generate interrupts
```

```
	config_timer();
	enable_nios2_interrupts(); // enable interrupts in the Nios II processor
	*LEDR_ptr=0;
	while (1){
		*LEDR_ptr=COUNT;
	} // wait for an interrupt
}
/* Set up the pushbutton KEYs port in the FPGA */
void config_KEYs(void){
	*(KEY_ptr + 2)=0b01000;		//KEY3사용
}
/* Enable interrupts in the Nios II processor */
void enable_nios2_interrupts(void){
	NIOS2_WRITE_IENABLE(0b011);	//IRQ pushbutton,TIMER
	NIOS2_WRITE_STATUS(0b01);		//interrupt.PIE=1
}
void config_timer(void){
	int pl=25000000;//0.25초
	int ph=pl>>16;
	*(TIMER_ptr+2)=pl;
	*(TIMER_ptr+3)=ph;
	*(TIMER_ptr+1)=0b0111; //stop,start,cont,ito=0111
}
//Lab7_Part2_exception_handler
#include "nios2_ctrl_reg_macros.h"
/* function prototypes */
void main(void);
void interrupt_handler(void);
void pushbutton_ISR(void);
void timer_ISR(void);
/* The assembly language code below handles Nios II reset processing */
void the_reset (void) __attribute__ ((section (".reset")));
void the_reset (void)
/*****************************************************************************
* Reset code; by using the section attribute with the name ".reset" we allow the linker program
* to locate this code at the proper reset vector address. This code just calls the main program
*****************************************************************************/
{
asm (".set noat"); // magic, for the C compiler
asm (".set nobreak"); // magic, for the C compiler
asm ("movia r2, main"); // call the C language main program
asm ("jmp r2");
}

/* The assembly language code below handles Nios II exception processing. This code should not be
```

```
* modified; instead, the C language code in the function interrupt_handler() can be modified as
* needed for a given application. */
void the_exception (void) __attribute__ ((section (".exceptions")));
void the_exception (void)
/*******************************************************************************
* Exceptions code; by giving the code a section attribute with the name ".exceptions" we allow
* the linker to locate this code at the proper exceptions vector address. This code calls the
* interrupt handler and later returns from the exception.
*******************************************************************************/
{
asm (".set noat"); // magic, for the C compiler
asm (".set nobreak"); // magic, for the C compiler
asm ( "subi sp, sp, 128");
asm ( "stw et, 96(sp)");      //r24
asm ( "rdctl et, ctl4");      //et=ipendig
asm ( "beq et, r0, SKIP_EA_DEC"); // interrupt is not external(SW interrupt)
asm ( "subi ea, ea, 4"); /* must decrement ea by one instruction for external
* interrupts, so that the instruction will be run(HW interrupt이기 때문에 pc-4) */
asm ( "SKIP_EA_DEC:" );
asm (    "stw      r1,   4(sp)" );                    // Save all registers
asm (    "stw      r2,   8(sp)" );
asm (    "stw      r3,   12(sp)" );
asm (    "stw      r4,   16(sp)" );
asm (    "stw      r5,   20(sp)" );
asm (    "stw      r6,   24(sp)" );
asm (    "stw      r7,   28(sp)" );
asm (    "stw      r8,   32(sp)" );
asm (    "stw      r9,   36(sp)" );
asm (    "stw      r10, 40(sp)" );
asm (    "stw      r11, 44(sp)" );
asm (    "stw      r12, 48(sp)" );
asm (    "stw      r13, 52(sp)" );
asm (    "stw      r14, 56(sp)" );
asm (    "stw      r15, 60(sp)" );
asm (    "stw      r16, 64(sp)" );
asm (    "stw      r17, 68(sp)" );
asm (    "stw      r18, 72(sp)" );
asm (    "stw      r19, 76(sp)" );
asm (    "stw      r20, 80(sp)" );
asm (    "stw      r21, 84(sp)" );
asm (    "stw      r22, 88(sp)" );
asm (    "stw      r23, 92(sp)" );      // r24 = et, 앞에서 이미 store함
asm (    "stw      r25, 100(sp)" );     // r25 = bt (skip r24 = et, because it is saved above)
asm (    "stw      r26, 104(sp)" );     // r26 = gp
// skip r27 because it is sp, and there is no point in saving this
```

```
asm (    "stw      r28, 112(sp)" );      // r28 = fp
asm (    "stw      r29, 116(sp)" );      // r29 = ea
asm (    "stw      r30, 120(sp)" );      // r30 = ba
asm (    "stw      r31, 124(sp)" );      // r31 = ra
asm (    "addi     fp,   sp, 128" );
asm (    "call interrupt_handler" ); // call the C language interrupt handler

asm (    "ldw      r1,   4(sp)" );             // Restore all registers
asm (    "ldw      r2,   8(sp)" );
asm (    "ldw      r3,   12(sp)" );
asm (    "ldw      r4,   16(sp)" );
asm (    "ldw      r5,   20(sp)" );
asm (    "ldw      r6,   24(sp)" );
asm (    "ldw      r7,   28(sp)" );
asm (    "ldw      r8,   32(sp)" );
asm (    "ldw      r9,   36(sp)" );
asm (    "ldw      r10, 40(sp)" );
asm (    "ldw      r11, 44(sp)" );
asm (    "ldw      r12, 48(sp)" );
asm (    "ldw      r13, 52(sp)" );
asm (    "ldw      r14, 56(sp)" );
asm (    "ldw      r15, 60(sp)" );
asm (    "ldw      r16, 64(sp)" );
asm (    "ldw      r17, 68(sp)" );
asm (    "ldw      r18, 72(sp)" );
asm (    "ldw      r19, 76(sp)" );
asm (    "ldw      r20, 80(sp)" );
asm (    "ldw      r21, 84(sp)" );
asm (    "ldw      r22, 88(sp)" );
asm (    "ldw      r23, 92(sp)" );
asm (    "ldw      r24, 96(sp)" );
asm (    "ldw      r25, 100(sp)" );      // r25 = bt
asm (    "ldw      r26, 104(sp)" );      // r26 = gp
// skip r27 because it is sp, and we did not save this on the stack
asm (    "ldw      r28, 112(sp)" );      // r28 = fp
asm (    "ldw      r29, 116(sp)" );      // r29 = ea
asm (    "ldw      r30, 120(sp)" );      // r30 = ba
asm (    "ldw      r31, 124(sp)" );      // r31 = ra
asm (    "addi     sp,   sp, 128" );
asm (    "eret" );
}
/******************************************************************************
* Interrupt Service Routine: Determines the interrupt source and calls the appropriate subroutine
******************************************************************************/
void interrupt_handler(void)
```

```
{
int ipending;
NIOS2_READ_IPENDING(ipending);
if ( ipending & 0x2 ) // pushbuttons are interrupt level 1
pushbutton_ISR( );//pushbutton에 의한 interrupt일 때
else if( ipending & 0x1 )
timer_ISR();//TIMER에 의한 interrupt일 때
// else, ignore the interrupt
return;
}
//Lab7_Part2_timer_ISR.c
#include "address_map_nios2.h"
extern volatile int *TIMER_ptr; // main.c 에서 선언했던 TIMER_BASE 에 대한 포인터 변수
extern int COUNT; // main.c 에서 선언했던 변수
extern int RUN;       // main.c 에서 선언했던 변수
void timer_ISR(void){
    if(RUN&*TIMER_ptr&0b1){//RUN이1이고 TO가 1일때
        *TIMER_ptr=0;//TO 초기화
        COUNT++;//COUNT함
    }
    return;
}
//Lab7_Part2_pushbutton_ISR.c
#include "address_map_nios2.h"
extern int RUN; //main code에서 사용한 전역변수 RUN
extern volatile int *KEY_ptr; //main code에서 사용한 전역변수
void pushbutton_ISR(void){
    int KEY_edge=*(KEY_ptr+3);//KEY edgecapture
    *(KEY_ptr+3)=KEY_edge;//edgecapture초기화
    if(KEY_edge==0b1000){//key3번이 눌리면
        RUN=~RUN;          //RUN 토글
    }
    return;
}
```

결과 및 토의

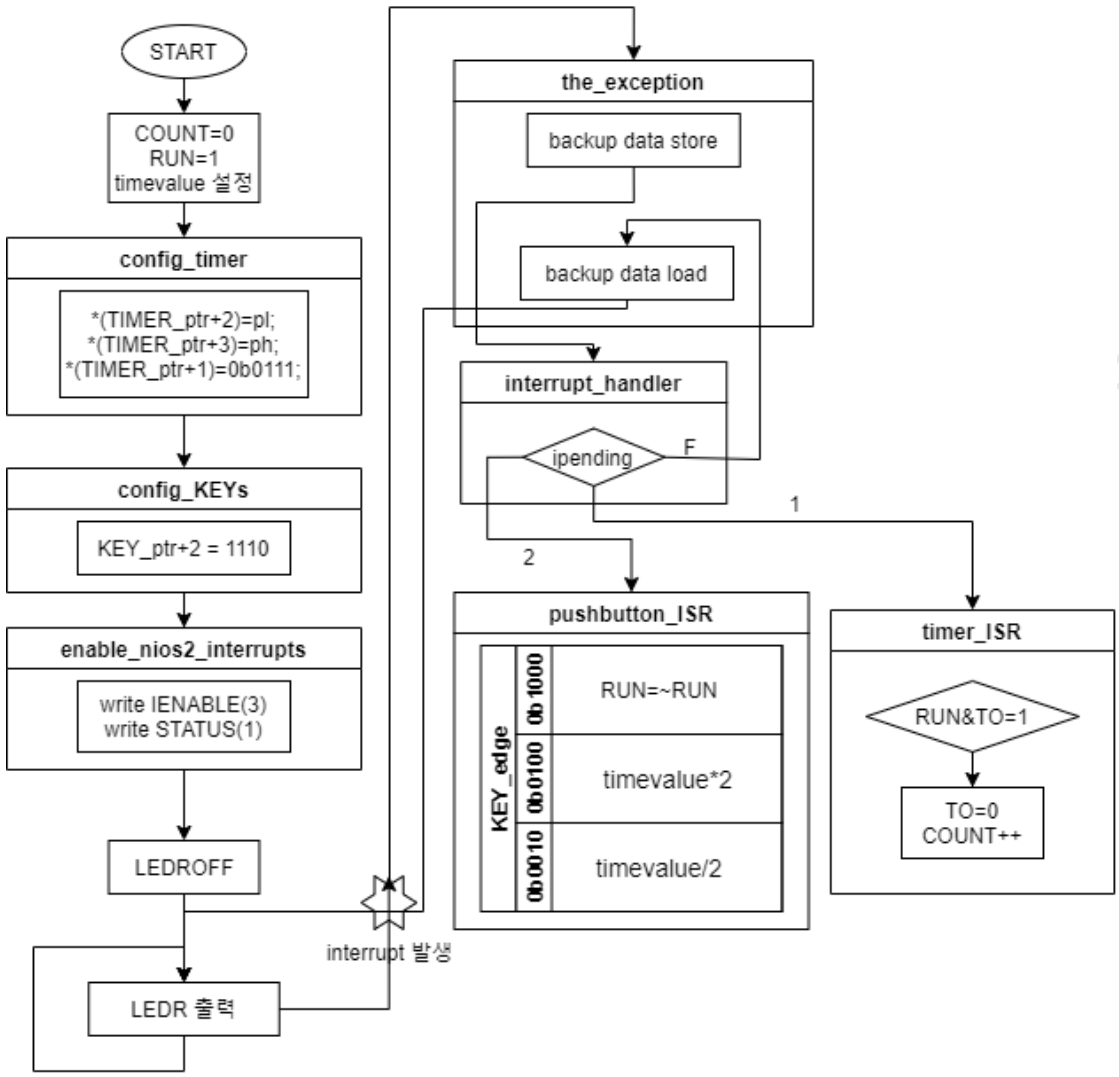
https://www.youtube.com/watch?v=8KxvuQ44hmY

KEY3을 누를 때 마다 LEDR로 출력되는 COUNT가 stop&start 하고 있는 것을 볼 수 있다.

전자 HW 설계 – 실습 보고서

✓ Part III

동작 원리.



Main code에서 config_KEYs와, config_timer, enable_nios2_interrups를 통해 key를 눌렀을 때와 timer가 0.25초 지날 때마다 interrupt가 발생해 supervisor모드에서 control 할 수 있게 설정한다. while문을 통해 지속적으로 LEDR 출력을 update할 수 있게 한다.

Exeption_handler code에서는 백업 데이터들(레지스터 값)을 모두 store한 후 ipending이 2이면(pushbutton에 의한 interrupt 의미)pushbutton_ISR을 call하고, 1이면 timer_ISR을 call한다. Timer_ISR과 pushbutton_ISR이 return 되면 다시 백업 데이터들을 load한다.

Pushbutton_ISR code에서는 KEY3이 눌릴 때 마다 RUN이 토글되게 한다. KEY2가 눌릴 때 마다 timevalue가 1/2배가 되도록 한다. KEY1가 눌릴 때 마다 timevalue가 2배가 되도록 한다.

Timer_ISR code에서는 RUN과 TO가 모두 1일 때 마다 TO를 초기화 해주고, COUNT한다.

구현 코드 설명

```
//Lab7_Part3_main.c
#include <stdio.h>
#include "nios2_ctrl_reg_macros.h"
#include "address_map_nios2.h"

int COUNT = 0; // global counter for red lights
int RUN = 1; // global, used to increment/not the count variable
int timevalue=25000000;//초기값 0.25
volatile int *KEY_ptr = (int*)KEY_BASE;
volatile int *LEDR_ptr = (int*)LEDR_BASE;
volatile int *TIMER_ptr = (int*)TIMER_BASE;//각각의 BASSE에 맞는 포인터
void enable_nios2_interrupts(void);
void config_KEYs(void);
void config_timer(void);
void main(void){
    config_KEYs(); // configure pushbutton KEYs to generate interrupts
    config_timer();
    enable_nios2_interrupts(); // enable interrupts in the Nios II processor
    *LEDR_ptr=0;
    while (1){
        *LEDR_ptr=COUNT;
    } // wait for an interrupt
}
/* Set up the pushbutton KEYs port in the FPGA */
void config_KEYs(void){
    *(KEY_ptr + 2)=0b01110;      //KEY3-1사용
}
/* Enable interrupts in the Nios II processor */
void enable_nios2_interrupts(void){
    NIOS2_WRITE_IENABLE(0b011);    //IRQ pushbutton
    NIOS2_WRITE_STATUS(0b01);       //interrupt.PIE=1
}
void config_timer(void){
    int pl=timevalue;//timevalue에 맞게 설정
    int ph=pl>>16;
    *(TIMER_ptr+2)=pl;
    *(TIMER_ptr+3)=ph;
    *(TIMER_ptr+1)=0b0111;//stop,start,cont,ito=0111
}
//Lab7_Part3_exception_handler
#include "nios2_ctrl_reg_macros.h"
/* function prototypes */
void main(void);
```

```
void interrupt_handler(void);
void pushbutton_ISR(void);
void timer_ISR(void);
/* The assembly language code below handles Nios II reset processing */
void the_reset (void) __attribute__ ((section (".reset")));
void the_reset (void)
/*****************************************************************************
* Reset code; by using the section attribute with the name ".reset" we allow the linker program
* to locate this code at the proper reset vector address. This code just calls the main program
*****************************************************************************/
{
asm (".set noat"); // magic, for the C compiler
asm (".set nobreak"); // magic, for the C compiler
asm ("movia r2, main"); // call the C language main program
asm ("jmp r2");
}

/* The assembly language code below handles Nios II exception processing. This code should not be
* modified; instead, the C language code in the function interrupt_handler() can be modified as
* needed for a given application. */
void the_exception (void) __attribute__ ((section (".exceptions")));
void the_exception (void)
/*****************************************************************************
* Exceptions code; by giving the code a section attribute with the name ".exceptions" we allow
* the linker to locate this code at the proper exceptions vector address. This code calls the
* interrupt handler and later returns from the exception.
*****************************************************************************/
{
asm (".set noat"); // magic, for the C compiler
asm (".set nobreak"); // magic, for the C compiler
asm ( "subi sp, sp, 128");
asm ( "stw et, 96(sp)");    //r24
asm ( "rdctl et, ctl4");    //et=ipendig
asm ( "beq et, r0, SKIP_EA_DEC"); // interrupt is not external(SW interrupt)
asm ( "subi ea, ea, 4"); /* must decrement ea by one instruction for external
* interrupts, so that the instruction will be run(HW interrupt이기 때문에 pc-4) */
asm ( "SKIP_EA_DEC:" );
asm (   "stw    r1,  4(sp)" );                  // Save all registers
asm (   "stw    r2,  8(sp)" );
asm (   "stw    r3,  12(sp)" );
asm (   "stw    r4,  16(sp)" );
asm (   "stw    r5,  20(sp)" );
asm (   "stw    r6,  24(sp)" );
asm (   "stw    r7,  28(sp)" );
asm (   "stw    r8,  32(sp)" );
```

```
asm (   "stw    r9,  36(sp)" );
asm (   "stw    r10, 40(sp)" );
asm (   "stw    r11, 44(sp)" );
asm (   "stw    r12, 48(sp)" );
asm (   "stw    r13, 52(sp)" );
asm (   "stw    r14, 56(sp)" );
asm (   "stw    r15, 60(sp)" );
asm (   "stw    r16, 64(sp)" );
asm (   "stw    r17, 68(sp)" );
asm (   "stw    r18, 72(sp)" );
asm (   "stw    r19, 76(sp)" );
asm (   "stw    r20, 80(sp)" );
asm (   "stw    r21, 84(sp)" );
asm (   "stw    r22, 88(sp)" );
asm (   "stw    r23, 92(sp)" );     // r24 = et, 앞에서 이미 store함
asm (   "stw    r25, 100(sp)" );    // r25 = bt (skip r24 = et, because it is saved above)
asm (   "stw    r26, 104(sp)" );    // r26 = gp
// skip r27 because it is sp, and there is no point in saving this
asm (   "stw    r28, 112(sp)" );    // r28 = fp
asm (   "stw    r29, 116(sp)" );    // r29 = ea
asm (   "stw    r30, 120(sp)" );    // r30 = ba
asm (   "stw    r31, 124(sp)" );    // r31 = ra
asm (   "addi   fp,  sp, 128" );
asm (   "call interrupt_handler" ); // call the C language interrupt handler

asm (   "ldw    r1,  4(sp)" );          // Restore all registers
asm (   "ldw    r2,  8(sp)" );
asm (   "ldw    r3,  12(sp)" );
asm (   "ldw    r4,  16(sp)" );
asm (   "ldw    r5,  20(sp)" );
asm (   "ldw    r6,  24(sp)" );
asm (   "ldw    r7,  28(sp)" );
asm (   "ldw    r8,  32(sp)" );
asm (   "ldw    r9,  36(sp)" );
asm (   "ldw    r10, 40(sp)" );
asm (   "ldw    r11, 44(sp)" );
asm (   "ldw    r12, 48(sp)" );
asm (   "ldw    r13, 52(sp)" );
asm (   "ldw    r14, 56(sp)" );
asm (   "ldw    r15, 60(sp)" );
asm (   "ldw    r16, 64(sp)" );
asm (   "ldw    r17, 68(sp)" );
asm (   "ldw    r18, 72(sp)" );
asm (   "ldw    r19, 76(sp)" );
asm (   "ldw    r20, 80(sp)" );
```

```
asm (   "ldw    r21, 84(sp)" );
asm (   "ldw    r22, 88(sp)" );
asm (   "ldw    r23, 92(sp)" );
asm (   "ldw    r24, 96(sp)" );
asm (   "ldw    r25, 100(sp)" );    // r25 = bt
asm (   "ldw    r26, 104(sp)" );    // r26 = gp
// skip r27 because it is sp, and we did not save this on the stack
asm (   "ldw    r28, 112(sp)" );    // r28 = fp
asm (   "ldw    r29, 116(sp)" );    // r29 = ea
asm (   "ldw    r30, 120(sp)" );    // r30 = ba
asm (   "ldw    r31, 124(sp)" );    // r31 = ra
asm (   "addi   sp,  sp, 128" );
asm (   "eret" );
}
/*****************************************************************************
* Interrupt Service Routine: Determines the interrupt source and calls the appropriate subroutine
*****************************************************************************/
void interrupt_handler(void)
{
int ipending;
NIOS2_READ_IPENDING(ipending);
if ( ipending & 0x2 ) // pushbuttons are interrupt level 1
pushbutton_ISR( );//pushbutton에 의한 interrupt일 때
else if( ipending & 0x1 )
timer_ISR();//TIMER에 의한 interrupt일 때
// else, ignore the interrupt
return;
}
//Lab7_Part3_timer_ISR
#include "address_map_nios2.h"
extern volatile int *TIMER_ptr; // main.c 에서 선언했던 TIMER_BASE 에 대한 포인터 변수
extern int COUNT; // main.c 에서 선언했던 변수
extern int RUN;     // main.c 에서 선언했던 변수
void timer_ISR(void){
    if(RUN&*TIMER_ptr&0b1){//RUN이1이고 TO가 1일때
        *TIMER_ptr=0;   //TO 초기화
        COUNT++;        //COUNT함
    }
    return;
}
Lab7_Part3_pushbutton_ISR.c
#include "address_map_nios2.h"
extern int RUN;                 //main code에서 사용한 전역변수 RUN
extern volatile int *KEY_ptr;   //main code에서 사용한 전역변수
extern volatile int *TIMER_ptr; //main code에서 사용한 전역변수
```

```
extern int timevalue;            //main code에서 사용한 전역변수
void pushbutton_ISR(void){
    int KEY_edge=*(KEY_ptr+3);  //KEY edgecapture
    *(KEY_ptr+3)=KEY_edge;       //KEY edgecapture 초기화
    if(KEY_edge==0b1000){        //KEY 3번이 눌리면
        RUN=~RUN;                //RUN 토글
    }
    else if(KEY_edge==0b0100){  //KEY 2번이 눌리면
        timevalue=timevalue*2;  //timevalue두배
        *(TIMER_ptr+2)=timevalue;       //변경된 timevalue에 맞게 timer재설정
        *(TIMER_ptr+3)=timevalue>>16;
        *(TIMER_ptr+1)=0b0111;
    }
    else if(KEY_edge==0b0010){ //KEY 1번이 눌리면
        timevalue=timevalue/2;  //timevalue0.5배
        *(TIMER_ptr+2)=timevalue;   //변경된 timevalue에 맞게 timer재설정
        *(TIMER_ptr+3)=timevalue>>16;
        *(TIMER_ptr+1)=0b0111;
    }
    return;
}
```

결과 및 토의

https://www.youtube.com/watch?v=nbFzBClDUrA

KEY3을 누를 때는 Part2와 동일하게 stop&start를 수행하고, KEY2를 누를 때 마다 COUNT속도가 2배씩 느려 지고, KEY1을 누를 때 마다 COUNT속도가 2배씩 빨라지는 것을 볼 수 있다.