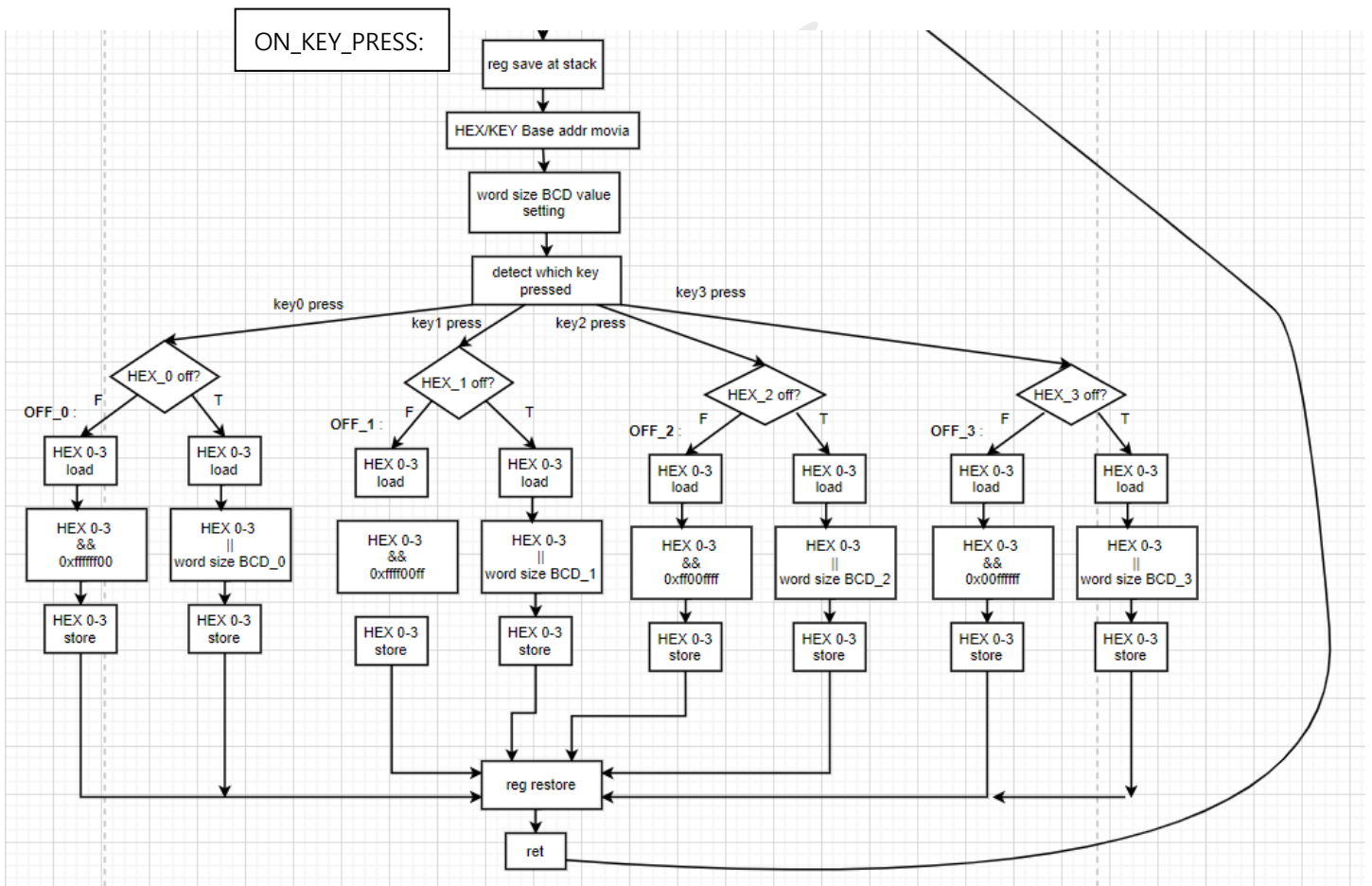
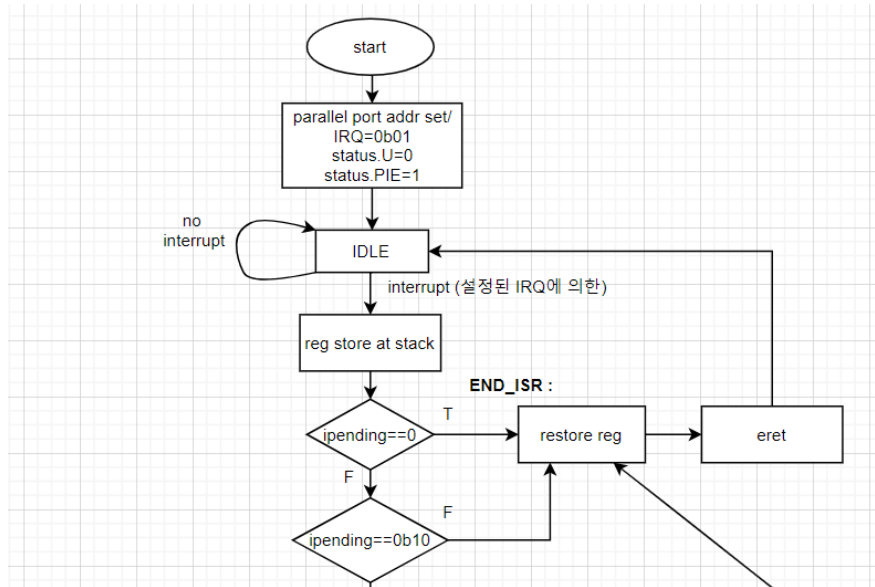


# 전자 HW 설계 - 실습 보고서

이름: 박관영 (2016124099)

## ✓ Part I

### 동작 원리



<main.s>

Nios II processor를 exception을 처리할 수 있도록 special purpose register들을 설정한다. Status.U=0, status.PIE=1로 하여 supervisor mode에서 동작하며 exception을 처리할 수 있는 상태로 만든다. Exception을 통해 조절할 peripherals의 base

# 전자 HW 설계 – 실습 보고서

address를 저장하고, 그 register map에 적절한 비트를 설정해 준다. 여기서는 key를 통해 interrupt을 발생시킬 것이므로 key의 interrupt mask를 0b1111로 설정하였다. 즉, 모든 키에 대해서 interrupt를 발생시킬 수 있게 하여 ISR로 진입할 수 있게 한다. 그 이후 IDLE 상태에 머물면서 interrupt이 발생하는 것을 감지한다.

<service.s>

ISR 모드에서 사용할 register들을 stack에 저장하고, 다시 restore하면서 eret instruction을 통해 interrupt을 handling한 후 main으로 돌아오게 된다. lpending reg를 통해 sw interrupt인지 hw interrupt인지 판별 후 interrupt handling을 진행한다. 만약 hw interrupt라면 해당 instruction이 제대로 수행되지 않았다고 볼 수 있으므로, ea=ea-4를 통해 exception handling 후 다시 수행할 수 있게 한다. Key에 의해 interrupt이 발생했다면 key\_isr을 call하여 key에 대한 interrupt를 수행하게 된다.

<key\_isr.s>

Key edge reg를 통해 어떤 키에 의해 interrupt이 발생하였는지 detect한 후 각 key에 적절한 subroutine을 수행하게 된다. 일단 key에 의해 interrupt이 발생하였다면 다음 interrupt이 발생하기 전까지는 IDLE 상태에 머물러야 하므로 key edge bit에 값을 stwio하여 IRQ를 turn off하게 된다. 그 후 Display\_0~3, OFF\_0~3을 수행하게 된다.

<Display>

해당 HEX device index에 해당하는 byte값을 읽어 해당 HEX device가 on 되어있는 지를 판단한다. on되어있다면 OFF subroutine으로 이동하고, off되어있다면 다른 HEX device에 영향을 주지 않고 해당 HEX device에만 on 동작을 수행하기 위해 or 연산을 사용하였다. HEX0-3 reg에 저장되어 있는 값을 word 단위로 읽고, word size의 bcd 값과 원래 HEX reg에 저장되어 있던 값을 or연산 취하여 해당 HEX만 on할 수 있게 된다.

<OFF>

Display와 반대로 해당 HEX만 off하는 subroutine이다. 각 device에 적절한 immediate를 설정한다. 즉 HEX0-3에서 각 device에 해당하는 byte만 0으로 설정되어 있는 immediate를 설정한다. 그 후 HEX0-3에 있는 value와 immediate를 and 연산하여 해당 device만 off할 수 있게 된다.

## 구현 코드 설명

```
.text
.global _start
_start:
    movia sp, 0x20000    ## stack set at dram
    movia r22, 0xFF200020    ## hex base addr
    stwio r0, (r22)

    movia r22, 0xFF200050    ## key base addr
    movi r23, 0b01111    ## set all interrupt mask bit 1
    stwio r23, 8(r22)

    movi r23, 2
    wrctl ienable, r23    ## set interrupt enable bit 1 (key interrupt)
```

# 전자 HW 설계 – 실습 보고서

```
movi r23, 1
wrctl status, r23      ## turn on Nios II interrupt processing
```

IDLE: br IDLE

```
ISR: subi sp, sp, 16    ## et, ea, ra, r22 will be stored stack
    stw et, (sp)
    rdctl et, ctl4      ## read ipending register
    beq et, r0, SKIP_EA_DEC  ## if hw interrupt occur
    subi ea, ea, 4      ## to handle exception occur reason
```

```
SKIP_EA_DEC: stw ea, 4(sp)
            stw ra, 8(sp)
                stw r22, 12(sp)
                rdctl et, ctl4
                bne et, r0, ON_EXT_INT
```

ON\_TRAP: br END\_ISR ## not define sw interrupt

```
ON_EXT_INT: andi r22, et, 0b10
            beq r22, r0, END_ISR  # no interrupt
            call ON_KEY_PRESS
```

```
END_ISR: ldw et, (sp)
        ldw ea, 4(sp)
        ldw ra, 8(sp)
        ldw r22, 12(sp)
        addi sp, sp, 16    ## reg restore
        eret
```

.global ON\_KEY\_PRESS

```
ON_KEY_PRESS:      subi sp, sp, 16    ## save r22, r23, r21, ra at stack
                    stw r22, (sp)
                    stw r23, 4(sp)
                    stw r21, 8(sp)
                    stw ra, 12(sp)

                    movia r22, 0xFF200020    ## hex base
                    movia r21, 0xFF200050    ## key base
                    ldwio r23, 0xC(r21)
                    stwio r23, 0xC(r21)      ## Turn off IRQ

                    movia r10, 0x3f          ## word size bcd value
                    movia r11, 0x0600
                    movia r12, 0x5b0000
                    movia r13, 0x4f000000
```

## 전자 HW 설계 – 실습 보고서

```
andi r4, r23, 0x01    ## detect which key pressed
bne r4, r0, Display_0
srli r23, r23, 0x01
andi r4, r23, 0x01
bne r4, r0, Display_1
srli r23, r23, 0x01
andi r4, r23, 0x01
bne r4, r0, Display_2
srli r23, r23, 0x01
andi r4, r23, 0x01
bne r4, r0, Display_3
```

ON\_KEY\_PRESS\_END:

```
ldw r22, (sp)    ## reg restore
ldw r23, 4(sp)
ldw r21, 8(sp)
ldw ra, 12(sp)
addi sp, sp, 16
ret
```

Display\_0:

```
ldbio r7, (r22)
bne r7, r0, OFF_0    ## detect hex0 on/off
ldwio r7,(r22)        ## reload word
or r7, r7, r10        ## to keep other hex digits
stwio r7,(r22)
br ON_KEY_PRESS_END
```

Display\_1:

```
ldbio r7, 1(r22)
bne r7, r0, OFF_1    ## detect hex1 on/off
ldwio r7,(r22)        ## reload word
or r7, r7, r11        ## to keep other hex digits
stwio r7,(r22)
br ON_KEY_PRESS_END
```

Display\_2:

```
ldbio r7, 2(r22)
bne r7, r0, OFF_2    ## detect hex2 on/off
ldwio r7,(r22)        ## reload word
or r7, r7, r12        ## to keep other hex digits
stwio r7,(r22)
br ON_KEY_PRESS_END
```

Display\_3:

```
ldbio r7, 3(r22)
bne r7, r0, OFF_3    ## detect hex3 on/off
```

전자 HW 설계 – 실습 보고서

```
ldwio r7,(r22)      ## reload word
or r7, r7, r13      ## to keep other hex digits
stwio r7,(r22)
br ON_KEY_PRESS_END

OFF_0: ldwio r9, (r22)
movia r8, 0xfffff00  ## off hex0/keep other hex digits
and r9,r9,r8
stwio r9, (r22)
br ON_KEY_PRESS_END

OFF_1: ldwio r9, (r22)
movia r8,0xffff00ff  ## off hex1/keep other hex digits
and r9,r9,r8
stwio r9, (r22)
br ON_KEY_PRESS_END

OFF_2: ldwio r9, (r22)
movia r8,0xff00ffff  ## off hex2/keep other hex digits
and r9,r9,r8
stwio r9, (r22)
br ON_KEY_PRESS_END

OFF_3: ldwio r9, (r22)
movia r8,0x00fffff  ## off hex3/keep other hex digits
and r9,r9,r8
stwio r9, (r22)
br ON_KEY_PRESS_END

.end
```

결과 및 토의

<main.s>

<table><tr><td>r21</td><td>0x00000000</td></tr><tr><td>r22</td><td>0xFF200020</td></tr><tr><td>r23</td><td>0x00000001</td></tr><tr><td>et</td><td>0x00000000</td></tr><tr><td>bt</td><td>0xFFFFFFFF</td></tr><tr><td>gp</td><td>0x00000000</td></tr><tr><td>sp</td><td>0x00020000</td></tr><tr><td>fp</td><td>0x00000000</td></tr><tr><td>ea</td><td>0x00000000</td></tr><tr><td>ba</td><td>0xFFFFFFFF</td></tr><tr><td>ra</td><td>0x00000000</td></tr></table>	r21	0x00000000	r22	0xFF200020	r23	0x00000001	et	0x00000000	bt	0xFFFFFFFF	gp	0x00000000	sp	0x00020000	fp	0x00000000	ea	0x00000000	ba	0xFFFFFFFF	ra	0x00000000	<table><tr><td>r20</td><td>0x00000000</td></tr><tr><td>r21</td><td>0x00000000</td></tr><tr><td>r22</td><td>0xFF200050</td></tr><tr><td>r23</td><td>0x00000001</td></tr><tr><td>et</td><td>0x00000000</td></tr><tr><td>bt</td><td>0xFFFFFFFF</td></tr><tr><td>gp</td><td>0x00000000</td></tr><tr><td>sp</td><td>0x00020000</td></tr><tr><td>fp</td><td>0x00000000</td></tr><tr><td>ea</td><td>0x00000000</td></tr></table>	r20	0x00000000	r21	0x00000000	r22	0xFF200050	r23	0x00000001	et	0x00000000	bt	0xFFFFFFFF	gp	0x00000000	sp	0x00020000	fp	0x00000000	ea	0x00000000	<table><tr><td>r20</td><td>0x00000000</td></tr><tr><td>r21</td><td>0x00000000</td></tr><tr><td>r22</td><td>0xFF200050</td></tr><tr><td>r23</td><td>0x0000000F</td></tr><tr><td>et</td><td>0x00000000</td></tr><tr><td>bt</td><td>0xFFFFFFFF</td></tr><tr><td>gp</td><td>0x00000000</td></tr></table>	r20	0x00000000	r21	0x00000000	r22	0xFF200050	r23	0x0000000F	et	0x00000000	bt	0xFFFFFFFF	gp	0x00000000	<table><tr><td>status</td><td>0x00000001</td></tr><tr><td>estatus</td><td>0x00000000</td></tr><tr><td>bstatus</td><td>0xFFFFFFFF</td></tr><tr><td>ienable</td><td>0x00000002</td></tr></table>	status	0x00000001	estatus	0x00000000	bstatus	0xFFFFFFFF	ienable	0x00000002
r21	0x00000000																																																																		
r22	0xFF200020																																																																		
r23	0x00000001																																																																		
et	0x00000000																																																																		
bt	0xFFFFFFFF																																																																		
gp	0x00000000																																																																		
sp	0x00020000																																																																		
fp	0x00000000																																																																		
ea	0x00000000																																																																		
ba	0xFFFFFFFF																																																																		
ra	0x00000000																																																																		
r20	0x00000000																																																																		
r21	0x00000000																																																																		
r22	0xFF200050																																																																		
r23	0x00000001																																																																		
et	0x00000000																																																																		
bt	0xFFFFFFFF																																																																		
gp	0x00000000																																																																		
sp	0x00020000																																																																		
fp	0x00000000																																																																		
ea	0x00000000																																																																		
r20	0x00000000																																																																		
r21	0x00000000																																																																		
r22	0xFF200050																																																																		
r23	0x0000000F																																																																		
et	0x00000000																																																																		
bt	0xFFFFFFFF																																																																		
gp	0x00000000																																																																		
status	0x00000001																																																																		
estatus	0x00000000																																																																		
bstatus	0xFFFFFFFF																																																																		
ienable	0x00000002																																																																		
>>> HEX base addr	>>> KEY base addr	>>> interrupt mask all bit 1	>>> ienable=0b10 (key interrupt), status.U=0, status.PIE=1																																																																

# 전자 HW 설계 – 실습 보고서

<service.s>

sp	0x0001FFD0	et	0x00000002
fp	0x00000000	bt	0xFFFFFFFF
ea	0x00000000	gp	0x00000000
ba	0xFFFFFFFF	sp	0x0001FFD0
ra	0x00000000	fp	0x00000000
		ea	0x00000000
		ba	0xFFFFFFFF
		ra	0x00000000
		status	0x00000000
		estatus	0x00000000
		bstatus	0xFFFFFFFF
		ienable	0x00000002
		ipending	0x00000002
>>> sp 감소하여 공간 save		>> ipending read	



<ON\_KEY\_PRESS>

r10	0x0000003F
r11	0x00000600
r12	0x005B0000
r13	0x4F000000

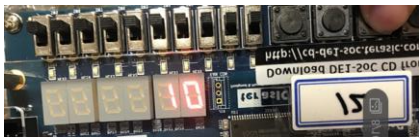

r23	0x00000001
-----	------------



>>> word size BCD value



>>> Stwio at edge bit → IRQ turn off

HEX0 on	HEX0 off												
<table> <tr><td>r7</td><td>0x00000000</td></tr> <tr><td>r7</td><td>0x0000003F</td></tr> </table> 	r7	0x00000000	r7	0x0000003F	<table> <tr><td>r8</td><td>0xFFFFFFFF</td></tr> <tr><td>r9</td><td>0x0000003F</td></tr> <tr><td>r8</td><td>0xFFFFFFFF</td></tr> <tr><td>r9</td><td>0x00000000</td></tr> </table> 	r8	0xFFFFFFFF	r9	0x0000003F	r8	0xFFFFFFFF	r9	0x00000000
r7	0x00000000												
r7	0x0000003F												
r8	0xFFFFFFFF												
r9	0x0000003F												
r8	0xFFFFFFFF												
r9	0x00000000												
R7을 통해 hex0에 word size bcd code0 전달한다.	해당 HEX device의 value가 0인 immediate와 and연산을 취하여 해당 HEX 를 off한다. 해당 비트를 제외한 다른 비트는 모두 1이므로 다른 HEX 에는 영향을 주지 않는다. (r9)를 통해 결과 값을 stwio한다.												

# 저자 NW 설계 - 실습 보고서

HEX1 on	HEX1 off
<div> <div>r7</div> <div>0x0000063F</div> </div> 	<div> <div>r7</div> <div>0x0000063F</div> </div> <div>→</div> <div> <div>r8</div> <div>0xFFFF00FF</div> </div> <div> <div>r9</div> <div>0x0000003F</div> </div> 
<p>R7을 통해 hex0에 bcd code1 전달 Bcd code가 word size이므로 HEX0에 영향을 주지않고 값을 display할 수 있는 것을 볼 수 있다. or연산을 통해서 stwio할 수 있다. R7을 통해 stwio 한다.</p>	<p>해당 HEX device의 value가 0인 immediate와 and연산을 취하여 해당 HEX 를 off한다. 해당 비트를 제외한 다른 비트는 모두 1이므로 다른 HEX에는 영향을 주지 않는다. (r9)를 통해 결과 값을 stwio한다.</p>

HEX2 on	HEX2 off
<div> <div>r7</div> <div>0x005B063F</div> </div> 	<div> <div>r7</div> <div>0x005B063F</div> </div> <div>→</div> <div> <div>r8</div> <div>0xFF00FFFF</div> </div> <div> <div>r9</div> <div>0x0000063F</div> </div> 
<p>R7을 통해 hex0에 bcd code2 전달 Bcd code가 word size이므로 HEX0에 영향을 주지않고 값을 display할 수 있는 것을 볼 수 있다. or연산을 통해서 stwio할 수 있다. R7을 통해 stwio 한다.</p>	<p>해당 HEX device의 value가 0인 immediate와 and연산을 취하여 해당 HEX 를 off한다. 해당 비트를 제외한 다른 비트는 모두 1이므로 다른 HEX에는 영향을 주지 않는다. (r9)를 통해 결과 값을 stwio한다.</p>

HEX3 on	HEX3 off
<div> <div>r7</div> <div>0x4F5B063F</div> </div> 	<div> <div>r7</div> <div>0x4F5B063F</div> </div> <div>→</div> <div> <div>r8</div> <div>0x00FFFFFF</div> </div> <div> <div>r9</div> <div>0x005B063F</div> </div> 
<p>R7을 통해 hex0에 bcd code1 전달 Bcd code가 word size이므로 HEX0에 영향을 주지않고 값을 display할 수 있는 것을 볼 수 있다. or연산을 통해서 stwio할 수 있다.</p>	<p>해당 HEX device의 value가 0인 immediate와 and연산을 취하여 해당 HEX 를 off한다. 해당 비트를 제외한 다른 비트는 모두 1이므로 다른 HEX에는 영향을 주지 않는다. (r9)를 통해 결과 값을 stwio한다.</p>

# 전자 HW 설계 - 실습 보고서

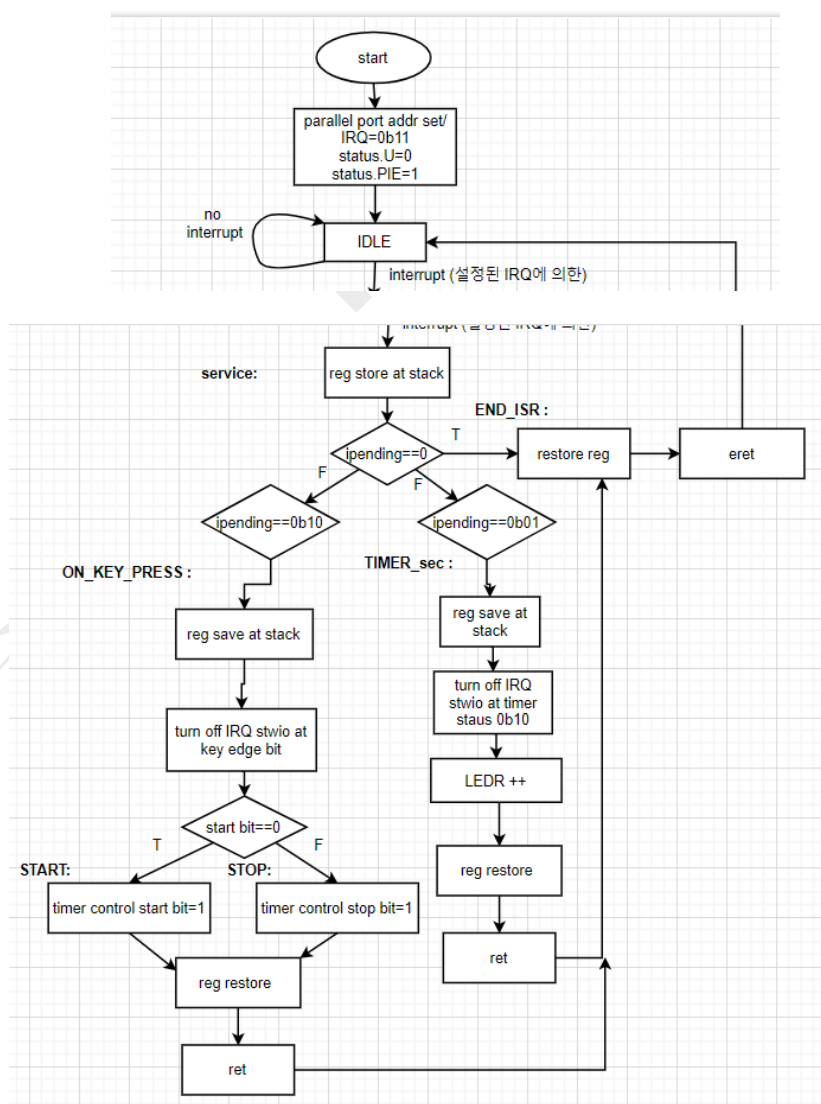
<ON\_KEY\_PRESS\_END>

r21	0x07208000
r22	0x00000002
r23	0x0B0D1317
et	0x00000002
bt	0xFFFFFFFF
gp	0x00000000
sp	0x00020000
fp	0x00000000
ea	0x0F4D0704
ba	0xFFFFFFFF
ra	0x00000058

Stack addr restore/  
other GPR restore

## ✓ Part II

동작 원리





# 전자 HW 설계 – 실습 보고서

<main.s>

사용할 parallel port의 base address를 저장한다. 여기서는 KEY, LEDR, timer peripherals를 사용할 것이므로 각 parallel port의 address를 저장하였다. 또한 NIOS II processor를 interrupt enable하게 하고, irq 1,2 즉, key, timer에 의해 interrupt를 처리할 수 있게 한다. 마지막으로 KEY, Timer가 interrupt를 발생할 수 있도록 register map을 설정한다. KEY의 interrupt mask를 0b1111로 설정하여 모든 KEY가 interrupt를 발생할 수 있게 하고, timer의 ITO bit를 on하여 Timer의 TO bit가 1이 될 때마다 interrupt를 발생하게 한다. 또한 timer의 initial status reg는 0b10으로 하고, control reg는 0b0111로 하여 timer가 연속적으로 동작하게 한다. Period는  $10^8$ 을 16bit씩 period low, high에 저장하여 real second를 측정할 수 있게 했다.

<service.s>

Part1의 service routine과 동일하나 sw interrupt와 hw interrupt의 구분만이 아니라 key와 timer의 interrupt를 구분해야 하므로, ipending reg의 bit를 관찰하여 둘 중 하나의 interrupt를 handling할 수 있게 하였다.

<key\_isr.s>

Ipending reg의 두번째 bit가 1일 경우 handling하게 되는 interrupt이다. Interrupt를 handling한 후 IDLE상태에 머물게 하기 위해 edge bit에 값을 stwio하여 IRQ를 turn off한다. 그 후 timer control register의 값을 읽어 start bit가 1인지 확인한다. 1이라면 stop subroutine을 수행하고, 0이라면 start subroutine을 수행한다. Start subroutine의 경우 control reg의 값을 0b0111로 update하고, stop subroutine의 경우 control reg의 값을 0b1011로 update한다. 그 후 sp와 reg를 restore하여 isr을 빠져나오게 된다. interrupt mask의 모든 bit가 1로 설정되어 있기 때문에 모든 key를 통해 stop/start를 toggle시킬 수 있게 된다.

<timer\_isr.s>

Ipending reg의 첫번째 bit가 1일 경우 handling하게 되는 interrupt이다. Interrupt를 handling한 후 IDLE상태에 머물게 하기 위해 IRQ를 turn off해야 한다. 이 때 timer register map의 경우에는 status를 다시 0b10으로 update하면 IRQ가 turn off된다. TO bit를 0으로 초기화 하여 다시 timer를 세게 되는 동안 interrupt가 발생하지 않으므로 IRQ가 off되는 것이다. 그 후 ledr에 표시될 비트를 +1 한다. Interrupt이 발생했다는 것은 timer의 TO bit가 1이 되었다는 것, 즉 1초를 센 것이므로 ledr에 표시될 비트를 하나 더해주어 초를 세게 된다. 그 후 reg restore한 뒤 isr을 빠져나오게 된다.

구현 코드 설명

```
.text
.global _start
_start:
    movia sp, 0x20000    ## stack set at dram
    movia r8, 0xFF200000 ## LEDR base addr
    movia r9, 0xFF202000 ## timer base addr
    movia r10, 0xFF200050 ## key base addr
    stwio r0, 0(r8)

    call CONFIG_TIMER
    call CONFIG_KEYS
```

## 전자 HW 설계 – 실습 보고서

```
movi r23, 3
wrctl ienable, r23    ## set interrupt enable key/timer
movi r23, 1
wrctl status, r23     ## turn on Nios II interrupt processing
```

IDLE: br IDLE

CONFIG\_TIMER:

```
movia r11, 0xe100      ## period_l setting
stwio r11, 8(r9)
movia r11, 0x05f5      ## period_h setting
stwio r11, 0xC(r9)

movi r11, 0b10         ## set timer status
stwio r11, 0(r9)

movi r11, 0b0111       ## set timer control interrupt enable
stwio r11, 4(r9)

ret
```

CONFIG\_KEYS:

```
movi r12, 0b1111      ## set all interrupt mask bit 1
stwio r12, 8(r10)
ret
```

.section .reset, "ax"

```
movia r2, _start
jmp r2
```

.section .exceptions, "ax"

.global ISR

```
ISR:    subi sp, sp, 16    ## et, ea, ra, r22 will be stored stack
        stw et, (sp)
        rdctl et, ctl4    ## read ipending register
        beq et, r0, SKIP_EA_DEC    ## if hw interrupt occur
        subi ea, ea, 4     ## to handle exception occur reason
```

```
SKIP_EA_DEC:    stw ea, 4(sp)
                stw ra, 8(sp)
                stw r22, 12(sp)
                rdctl et, ctl4
                bne et, r0, ON_EXT_INT
```

# 전자 HW 설계 – 실습 보고서

```
ON_TRAP: br END_ISR      ## not define sw interrupt

ON_EXT_INT:              andi r22, et, 0b01
                        beq  r22, r0, KEY_INT
                        call TIMER_sec

END_ISR: ldw et, (sp)
        ldw ea, 4(sp)
        ldw ra, 8(sp)
        ldw r22, 12(sp)
        addi sp, sp, 16    ## reg restore
        eret

KEY_INT: andi r22, et, 0b10
        beq r22, r0, END_ISR
        call ON_KEY_PRESS
        br END_ISR

.global ON_KEY_PRESS
ON_KEY_PRESS: subi sp, sp, 20    ## save reg at stack
        stw r22, (sp)

        stw r23, 4(sp)
        stw r21, 8(sp)
        stw r4, 12(sp)
        stw ra, 16(sp)

        movia r22, 0xFF200050    ## key base addr
        ldwio r23, 0xC(r22)      ## key edgebit
        stwio r23, 0xC(r22)      ## turn off IRQ

        movia r20, 0xFF202000    ## Timer base addr
        ldwio r23, 4(r20)        ## Timer control reg check

        andi  r21, r23, 0b0100
        beq  r21, r0, START      ## if TIMER not run
        br  STOP

ON_KEY_PRESS_END:

        ldw r22, (sp)            ## reg restore
        ldw r23, 4(sp)
        ldw r21, 8(sp)
        ldw r4, 12(sp)
```

## 전자 HW 설계 – 실습 보고서

```
        ldw ra, 16(sp)
        addi sp, sp, 20
        ret

START:

        movi r4, 0b0111      ## start bit on
        stwio r4, 4(r20)
        br ON_KEY_PRESS_END

STOP:    movi r4, 0b1011      ## stop bit on
        stwio r4, 4(r20)
        br ON_KEY_PRESS_END

.global TIMER_sec
TIMER_sec:
        subi sp, sp, 16      ## save reg at stack
        stw r20, (sp)
        stw r21, 4(sp)
        stw r22, 8(sp)
        stw ra, 12(sp)

        movia r20, 0xff202000 ## Timer base addr
        movia r21, 0xff200000 ## LEDR base addr

        movi r22, 0b10       ## turn off IRQ
        stwio r22, (r20)

        ldwio r22, (r21)
        addi r22, r22, 1      ## real second ++ at ledr
        stwio r22, (r21)

        ldw r20, (sp)        ## restore reg
        ldw r21, 4(sp)
        ldw r22, 8(sp)
        ldw ra, 12(sp)
        addi sp, sp, 16

        ret

.end
```

결과 및 토의

<main.s>

# 전자 HW 설계 – 실습 보고서

<div> <div>r8</div> <div>0xFF200000</div> </div> <div> <div>r9</div> <div>0xFF202000</div> </div> <div> <div>r10</div> <div>0xFF200050</div> </div>	<div> <div>sp</div> <div>0x00020000</div> </div>	<div> <div>r11</div> <div>0x0000E100</div> </div> <div> <div>r11</div> <div>0x00000000</div> </div> <div> <div>r11</div> <div>0x00005F5</div> </div>	<div> <div>r11</div> <div>0x00000002</div> </div> <div> <div>r11</div> <div>0x00000000</div> </div> <div> <div>r11</div> <div>0x00000007</div> </div>	<div> <div>r12</div> <div>0x0000000F</div> </div> <div> <div>r12</div> <div>0x00000000</div> </div>
LED, TIMER, KEY base addr	sp at dram	Period_l Period_h	Timer status reg Timer control reg	Key interrupt mask All bit 1

status	0x00000001
estatus	0x00000000
bstatus	0xFFFFFFFF
ienable	0x00000003
ipending	0x00000001

Status.U=0, status.PIE=1  
 lenable=0b11(key, timer interrupt),  
 ipending이 1번째 bit가 1이 된 것  
 을 통해 timer interrupt이 발생함  
 을 알 수 있다.

<service.s>

Part1과 같은 ISR 동작을 수행하게 하는 routine이다. Ipending의 bit value를 판단하는 것만 다르므로 ipending의 reg를 통  
해 둘 중 하나의 interrupt을 처리하는 동작을 수행한다.

<key\_isr.s>

ipending 00000002

Cpulator 사이트를 통한 ipending reg value이다. Monitor program에서는 debugger를 통해 key interrupt 시 reg를 확인할  
수 없었으나 보드에서 정상 동작됨은 확인하였다.

Ipending reg의 2번째 bit가 on되었을 때 수행되는 interrupt handling이다. 아무 키나 누를 때 timer의 start/stop이 toggle  
되도록 exception을 handling한다. Timer control reg의 값을 통해 start/stop routine을 실행한다.

<timer\_isr.s>

Ipending reg의 첫번째 bit가 on되었을 때 수행되는 interrupt handling이다. 매우 빠른 시간 내에 TO bit이 toggle되므로  
reg의 값을 확인할 수는 없었다. Sp를 사용하여 reg의 값을 save/restore하였으므로 LEDR에 표시되는 bit의 value또한  
debugger로 확인할 수는 없었다. 그러나 board에 표시되는 ledr의 bit를 통해 실험 결과가 제대로 나왔음을 확인하였다.

ipending 0x00000001

# 전자 HW 설계 - 실습 보고서

