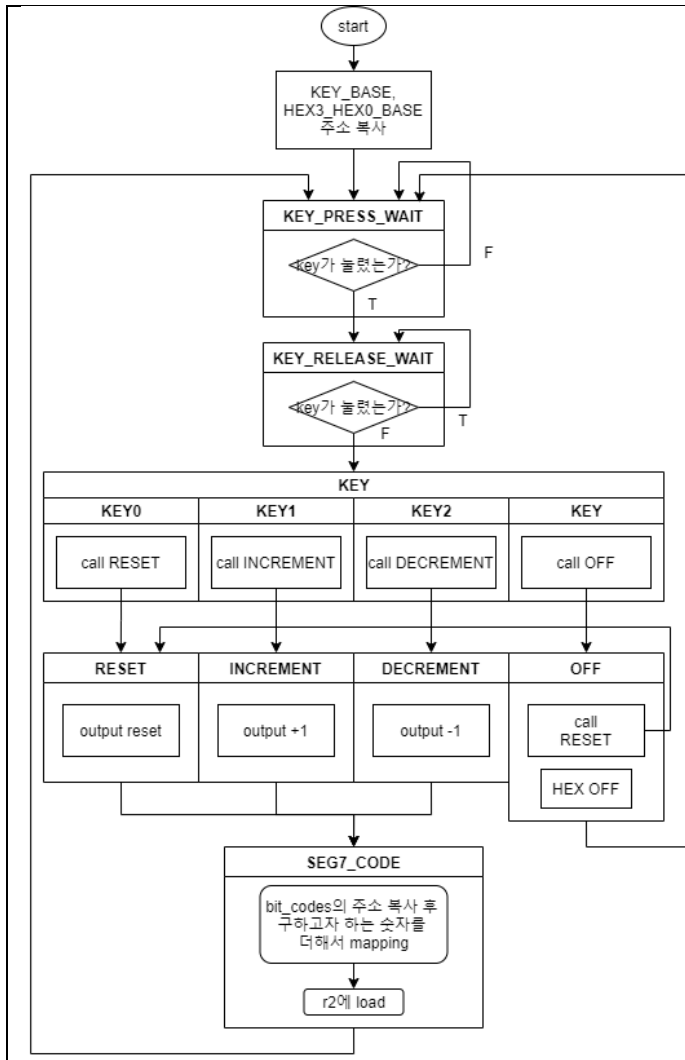


전자 HW 설계 – 실습 보고서

이름: 양해찬 (2016124145)

✓ Part I

동작 원리



Part1은 Polled IO를 이용해서 KEY3-0번을 누르면서 바뀌는 출력값을 HEX로 출력하는 것이다.

KEY0 : output을 0으로 reset 한다.

KEY1 : output을 1 증가시킨다.

KEY2 : output을 1 감소시킨다.

KEY3 : HEX를 OFF시킨다.

먼저 KEY가 눌렀다 떼어지면 KEY0-3까지 각자의 역할에 맞게 함수들이 호출되고 output이 바뀌게 된다.

바뀐 output값을 SEG7_CODE를 통해 (Lab2와 동일) HEX로 출력한다.

Output이 0미만이면 0을 유지하도록 했고,

Output이 9를 초과한 상태에서 다시 key1을 누르면 1부터 다시 출력하도록 설계했다.

구현 코드 설명

```
.include "address_map_nios2.s"
.text
.global _start

_start:  movia  r7, KEY_BASE      #r7에 KEY_BASE주소 복사
        movia  r8, HEX3_HEX0_BASE #r8에 HEX3_HEX0_BASE주소 복사
        movia  sp, 0x20000      #sp에 주소값으로 0x20000복사
        movi   r9, 9

RST:    mov    r3, r0            #r3(결과값) 초기화
```

전자 HW 설계 – 실습 보고서

```
MAIN:      blt    r3,  r0,  RST
           bgt    r3,  r9,  RST          #output이 0~9범위 밖이면 리셋
KEY_PRESS_WAIT: ldwio r6,  0(r7)          #r6에 key3~0상태 load
           beq    r6,  r0,  KEY_PRESS_WAIT #r6이 0이면 다시 KEY_PRESS_WAIT(아무것도 누르지 않았다면)
KEY_RELEASE_WAIT: ldwio r5,  0(r7)          #r6에 key3~0상태 load
           bne    r5,  r0,  KEY_RELEASE_WAIT #r6이 0이 아니면 다시 KEY_PRESS_WAIT(누른채로 떼지 않았다면)

           andi   r4,  r6,  0x1
           bne    r4,  r0,  RESET_CALL      #key0이 눌렀다면 RESET_CALL
           srli   r6,  r6,  0x1
           andi   r4,  r6,  0x1
           bne    r4,  r0,  INCREMENT_CALL  #key1이 눌렀다면 INCREMENT_CALL
           srli   r6,  r6,  0x1
           andi   r4,  r6,  0x1
           bne    r4,  r0,  DECREMENT_CALL  #key2이 눌렀다면 DECREMENT_CALL
           srli   r6,  r6,  0x1
           andi   r4,  r6,  0x1
           bne    r4,  r0,  OFF_CALL        #key3이 눌렀다면 OFF_CALL

INCREMENT_CALL:
           call   INCREMENT      #INCREMENT 호출
           call   SEG7_CODE      #SEG7_CODE 호출
           br     MAIN          #다시 MAIN으로

DECREMENT_CALL:
           call   DECREMENT
           call   SEG7_CODE
           br     MAIN

RESET_CALL:
           call   RESET
           call   SEG7_CODE
           br     MAIN

OFF_CALL:
           call   OFF
           br     MAIN

INCREMENT:  addi   r3,  r3,  1      #결과값 1증가
           bgt    r3,  r9,  RST    #10이상이면 다시 돌아감
           ret

DECREMENT:  subi   r3,  r3,  1      #결과값 1감소
           blt    r3,  r0,  RST    #0미만이면 다시 돌아감
           ret

RESET:      mov    r3,  r0          #결과값 초기화
           ret





OFF:        subi   sp,  sp,  4      #call된 서브루틴 내에서 다시call하므로 return할 주소를 저장해야한다
           stw     ra,  (sp)
```

전자 HW 설계 – 실습 보고서

```
call RESET
stwio r0, (r8)      #HEX OFF
ldw ra, (sp)        #저장해놓았던 return할 주소 다시 load
addi sp, sp, 4      #sp 원상복구
ret
```

```
BIT_CODES:          .byte 0b00111111,0b00000110,0b01011011,0b01001111,0b01100110
                    .byte 0b01101101,0b01111101,0b00000111,0b01111111,0b01100111
                    .skip 2 #숫자가 총 10개이므로 10byte이다. word단위로 만들어주기 위해 2byte skip
SEG7_CODE:          movia r15,BIT_CODES #BIT_CODES 주소 복사
                    add r15,r15,r3      #r3의 숫자에 맞게 주소값에 더해서 해당 bitcode 주소를 가질 수 있게한다.
                    ldb r2,(r15)        #r2에 load
                    stwio r2, (r8)      #r8에 store
                    ret                #return
```

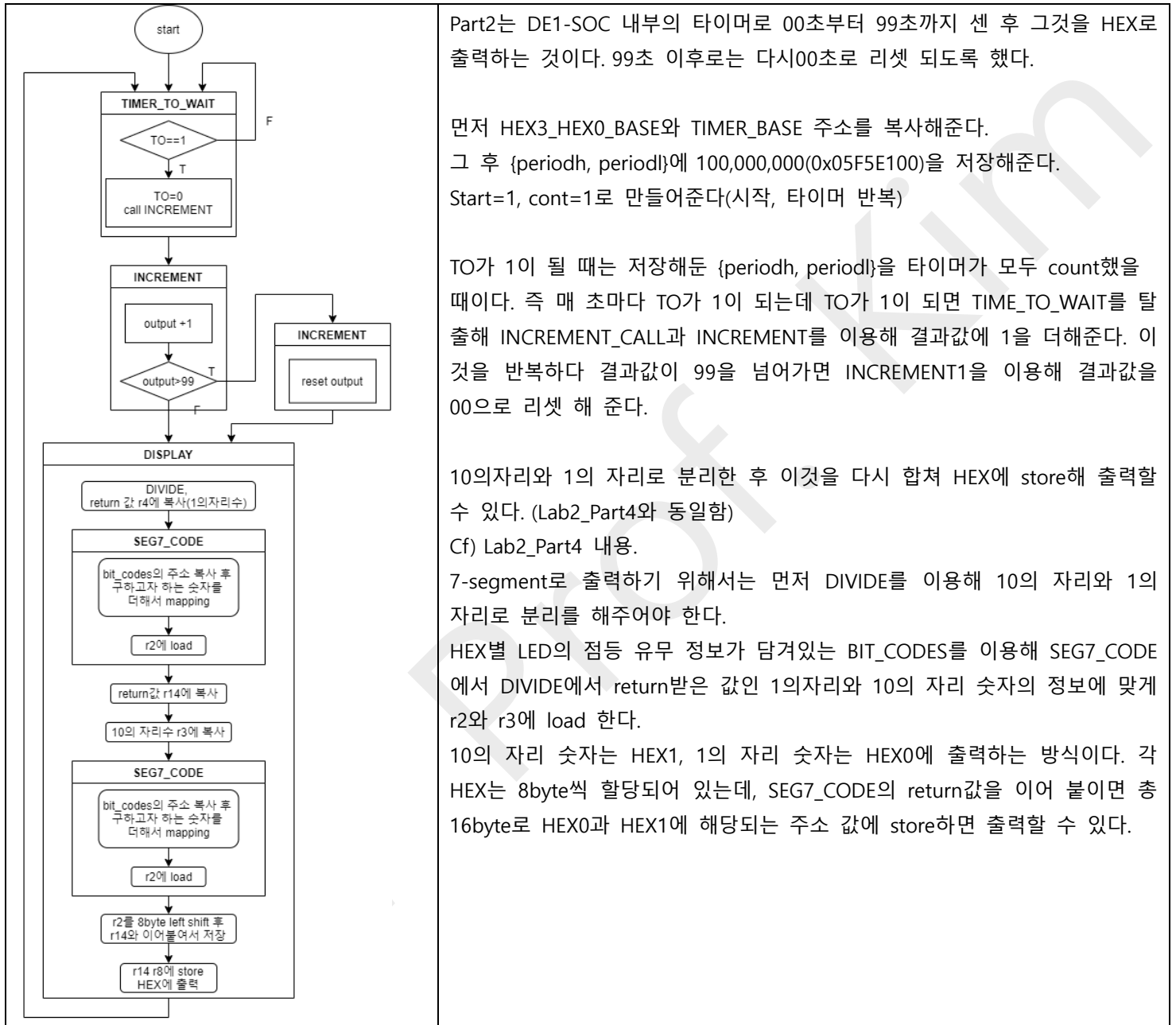
결과 및 토의

<table><tr><td>pc</td><td>0x0000002C</td><td>r19</td><td>0x00000000</td></tr><tr><td>zero</td><td>0x00000000</td><td>r20</td><td>0x00000000</td></tr><tr><td>r1</td><td>0x00000000</td><td>r21</td><td>0x00000000</td></tr><tr><td>r2</td><td>0x0000006D</td><td>r22</td><td>0x00000000</td></tr><tr><td>r3</td><td>0x00000005</td><td>r23</td><td>0x00000000</td></tr><tr><td>r4</td><td>0x00000001</td><td>et</td><td>0x00000000</td></tr><tr><td>r5</td><td>0x00000000</td><td>bt</td><td>0xFFFFFFFF</td></tr><tr><td>r6</td><td>0x00000000</td><td>gp</td><td>0x00000000</td></tr><tr><td>r7</td><td>0xFF200050</td><td>sp</td><td>0x00020000</td></tr><tr><td>r8</td><td>0xFF200020</td><td>fp</td><td>0x00000000</td></tr><tr><td>r9</td><td>0x00000009</td><td>ea</td><td>0x00000000</td></tr><tr><td>r10</td><td>0x00000000</td><td>ba</td><td>0xFFFFFFFF</td></tr><tr><td>r11</td><td>0x00000000</td><td>ra</td><td>0x0000006C</td></tr><tr><td>r12</td><td>0x00000000</td><td>status</td><td>0x00000000</td></tr><tr><td>r13</td><td>0x00000000</td><td>estatus</td><td>0x00000000</td></tr><tr><td>r14</td><td>0x00000000</td><td>bstatus</td><td>0xFFFFFFFF</td></tr><tr><td>r15</td><td>0x00000001</td><td>ienable</td><td>0x00000000</td></tr><tr><td>r16</td><td>0x00000000</td><td>ipending</td><td>0x00000000</td></tr><tr><td>r17</td><td>0x00000000</td><td>cpuid</td><td>0x00000000</td></tr><tr><td>r18</td><td>0x00000000</td><td></td><td></td></tr></table>	pc	0x0000002C	r19	0x00000000	zero	0x00000000	r20	0x00000000	r1	0x00000000	r21	0x00000000	r2	0x0000006D	r22	0x00000000	r3	0x00000005	r23	0x00000000	r4	0x00000001	et	0x00000000	r5	0x00000000	bt	0xFFFFFFFF	r6	0x00000000	gp	0x00000000	r7	0xFF200050	sp	0x00020000	r8	0xFF200020	fp	0x00000000	r9	0x00000009	ea	0x00000000	r10	0x00000000	ba	0xFFFFFFFF	r11	0x00000000	ra	0x0000006C	r12	0x00000000	status	0x00000000	r13	0x00000000	estatus	0x00000000	r14	0x00000000	bstatus	0xFFFFFFFF	r15	0x00000001	ienable	0x00000000	r16	0x00000000	ipending	0x00000000	r17	0x00000000	cpuid	0x00000000	r18	0x00000000				
pc	0x0000002C	r19	0x00000000																																																																															
zero	0x00000000	r20	0x00000000																																																																															
r1	0x00000000	r21	0x00000000																																																																															
r2	0x0000006D	r22	0x00000000																																																																															
r3	0x00000005	r23	0x00000000																																																																															
r4	0x00000001	et	0x00000000																																																																															
r5	0x00000000	bt	0xFFFFFFFF																																																																															
r6	0x00000000	gp	0x00000000																																																																															
r7	0xFF200050	sp	0x00020000																																																																															
r8	0xFF200020	fp	0x00000000																																																																															
r9	0x00000009	ea	0x00000000																																																																															
r10	0x00000000	ba	0xFFFFFFFF																																																																															
r11	0x00000000	ra	0x0000006C																																																																															
r12	0x00000000	status	0x00000000																																																																															
r13	0x00000000	estatus	0x00000000																																																																															
r14	0x00000000	bstatus	0xFFFFFFFF																																																																															
r15	0x00000001	ienable	0x00000000																																																																															
r16	0x00000000	ipending	0x00000000																																																																															
r17	0x00000000	cpuid	0x00000000																																																																															
r18	0x00000000																																																																																	
	Key0을 눌러 reset 한 상태	Key1을 세번 눌러 3 출력																																																																																
																																																																																		
	3인 상태에서 key2를 두 번 눌러 출력이 1	Key3을 눌러 HEX OFF																																																																																
	왼쪽 registers는 key1을 다섯번 누른 상태에서 정지했을 때의 상태이다. Key1을 다섯번 눌렀기 때문에 출력 값이 5인데 r3 레지스터에서 잘 출력되고 있다. R2가 0x6D인 이유는 현재 출력 값인 5에 대한 segment binary code가 0b0110110이기 때문이다.																																																																																	
Key1을 5번 눌렀을 때의 Registers																																																																																		

전자 HW 설계 – 실습 보고서

✓ Part II

동작 원리.



Part2는 DE1-SOC 내부의 타이머로 00초부터 99초까지 센 후 그것을 HEX로 출력하는 것이다. 99초 이후로는 다시 00초로 리셋 되도록 했다.

먼저 HEX3_HEX0_BASE와 TIMER_BASE 주소를 복사해준다.
그 후 {periodh, periodl}에 100,000,000(0x05F5E100)을 저장해준다.
Start=1, cont=1로 만들어준다(시작, 타이머 반복)

TO가 1이 될 때는 저장해둔 {periodh, periodl}을 타이머가 모두 count했을 때이다. 즉 매 초마다 TO가 1이 되는데 TO가 1이 되면 TIME_TO_WAIT를 탈출해 INCREMENT_CALL과 INCREMENT를 이용해 결과값에 1을 더해준다. 이것을 반복하다 결과값이 99를 넘어가면 INCREMENT1을 이용해 결과값을 00으로 리셋 해 준다.

10의자리와 1의 자리로 분리한 후 이것을 다시 합쳐 HEX에 store해 출력할 수 있다. (Lab2_Part4와 동일함)

Cf) Lab2_Part4 내용.

7-segment로 출력하기 위해서는 먼저 DIVIDE를 이용해 10의 자리와 1의 자리로 분리를 해주어야 한다.

HEX별 LED의 점등 유무 정보가 담겨있는 BIT_CODES를 이용해 SEG7_CODE에서 DIVIDE에서 return받은 값인 1의자리와 10의 자리 숫자의 정보에 맞게 r2와 r3에 load 한다.

10의 자리 숫자는 HEX1, 1의 자리 숫자는 HEX0에 출력하는 방식이다. 각 HEX는 8byte씩 할당되어 있는데, SEG7_CODE의 return값을 이어 붙이면 총 16byte로 HEX0과 HEX1에 해당되는 주소 값에 store하면 출력할 수 있다.

구현 코드 설명

```
.include "address_map_nios2.s"
```

```
.text
```

```
.global _start
```

```
_start:
```

```
movia r8, HEX3_HEX0_BASE #r8에 HEX3_HEX0_BASE주소 복사
```

```
movia r9, TIMER_BASE #r9에 TIMER_BASE주소 복사
```

전자 HW 설계 – 실습 보고서

```
movia sp, 0x20000 #sp에 주소값으로 0x20000복사
mov r7, r0 #결과값 초기화
movui r10, 0xE100
stwio r10, 8(r9)
movui r10, 0x05F5
stwio r10, 12(r9) #r9 {periodh, periodl}에 100,000,000 저장
movui r10, 0x06
stwio r10, 4(r9) #start=1 cont=1
movi r11, 0x063 #99
```

MAIN:

```
TIMER_TO_WAIT: ldwio r6, (r9)
                andi r6, r6, 0x01
                beq r6, r0, TIMER_TO_WAIT #TO가 1이 아니면 다시 TIMER_TO_WAIT
                stbio r0, (r9) #TO를 다시 0으로
                br INCREMENT_CALL #TO가 1이 될때마다 INCREMENT_CALL분기
```

INCREMENT_CALL:

```
call INCREMENT
call DISPLAY
br MAIN
```

```
INCREMENT: addi r7, r7, 1 #output+1
            bgt r7, r11, INCREMENT1 #output이 99보다 크면 INCREMENT1
            ret
```

```
INCREMENT1: mov r7, r0 #output 초기화
            ret
```

DIVIDE:

```
movi r5,10
movi r3,0 #십의 자리수
```

```
CONT: blt r2,r5,return #r2가 10보다 작으면 return, r2는 1의 자리 수
      sub r2,r2,r5 #r2에 10을 뺄 때마다 r3에 더함으로써 r3이 10의 자리수를 가질 수 있다.
      addi r3,r3,1
      br CONT
```

return: ret

```
BIT_CODES: .byte 0b00111111,0b00000110,0b01011011,0b01001111,0b01100110
            .byte 0b01101101,0b01111101,0b00000111,0b01111111,0b01100111
            .skip 2 #숫자가 총 10개이므로 10byte이다. word단위로 만들어주기 위해 2byte skip
```



```
SEG7_CODE: movia r15,BIT_CODES #BIT_CODES 주소 복사
            add r15,r15,r4 #r4의 숫자에 맞게 주소값에 더해서 해당 bitcode 주소를 가질 수 있게한다.
            ldb r2,(r15) #r2에 load
            ret
```

```
DISPLAY: subi sp, sp, 4 #call된 서브루틴 내부에서 다시 call하므로 return할 주소 저장
          stw ra, (sp)
```

전자 HW 설계 - 실습 보고서

mov	r2,	r7	#output r2로 복사
call	DIVIDE		#DIVIDE 호출
mov	r4,	r2	#r4에 r2 복사(1의 자리 수)
call	SEG7_CODE		#1의 자리 수에 맞게 mapping
mov	r14,	r2	#return받은 값을 r14에 복사
mov	r4,	r3	#r4에 r3 복사(10의 자리 수)
call	SEG7_CODE		#10의 자리 수에 맞게 mapping
slli	r2,	r2,	8
or	r14,	r14,	r2 #r2와 r3에 저장된 값을 r14에 하나로 이어서 저장
stwio	r14,	(r8)	#r8에 store(HEX에 출력)
ldw	ra,	(sp)	#return하기위해 저장했던 주소 load
addi	sp,	sp,	4 #sp 원상복구
ret			

결과 및 토의

pc	0x00000038	r19	0x00000000		
zero	0x00000000	r20	0x00000000	9초가 흐른 후	100초 일 때
r1	0x00000000	r21	0x00000000		
r2	0x00006000	r22	0x00000000	왼쪽 사진은 52초 일 때 정지한 registers 상태이다. Output 값인 52(0x34)가 r7에 잘 출력되고 있는 것을 볼 수 있다. R2는 5에 대한 segment binary code가 0b0110110이고, 그것을 left shift 한 0x6D00이다. R14는 0x6D00과 2에 대한 segment binary code 0b01011011을 이어 붙인 0x6D5B이다.	
r3	0x00000005	r23	0x00000000		
r4	0x00000005	et	0x00000000		
r5	0x0000000A	bt	0xFFFFFFFF		
r6	0x00000000	gp	0x00000000		
r7	0x00000034	sp	0x00020000		
r8	0xFF200020	fp	0x00000000		
r9	0xFF202000	ea	0x00000000		
r10	0x00000006	ba	0xFFFFFFFF		
r11	0x00000063	ra	0x00000054		
r12	0x00000000	status	0x00000000		
r13	0x00000000	estatus	0x00000000		
r14	0x00006D5B	bstatus	0xFFFFFFFF		
r15	0x00000080	ienable	0x00000000		
r16	0x00000000	ipending	0x00000000		
r17	0x00000000	cpuid	0x00000000		
r18	0x00000000				

52초 일 때 registers