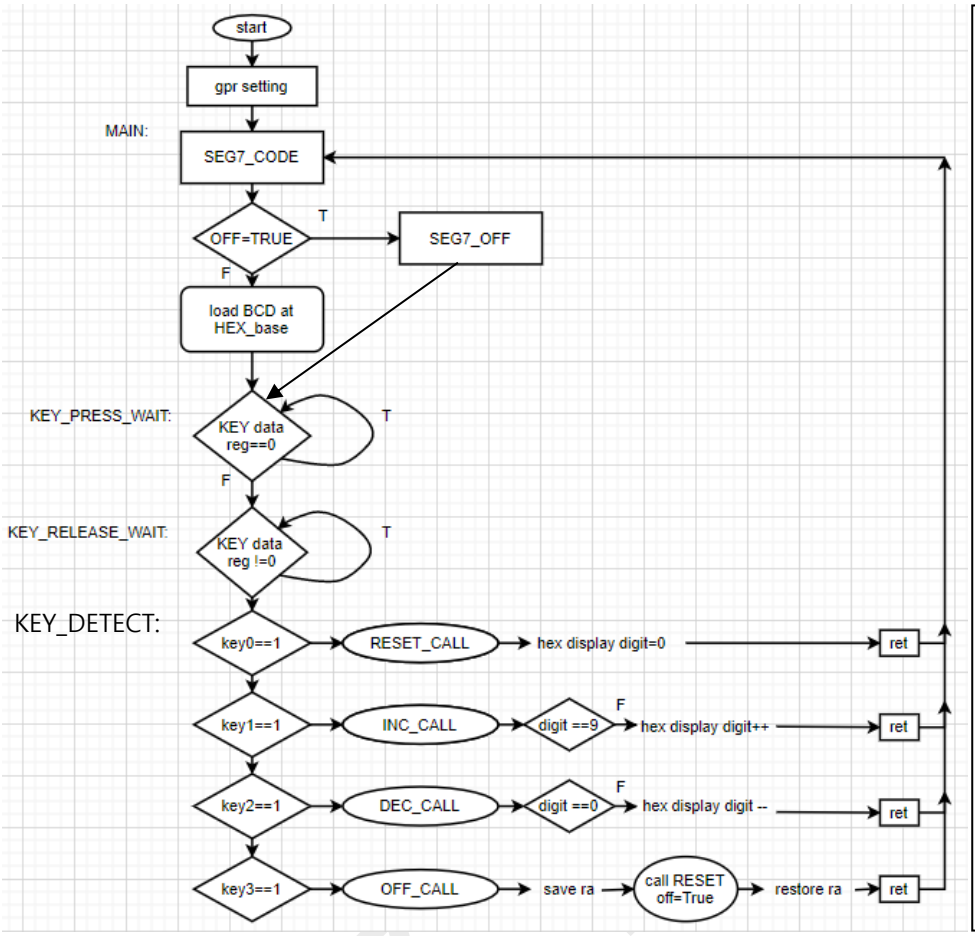
# 전자 HW 설계 – 실습 보고서

이름: 박관영 (2016124099)

## ✓ Part I

*동작 원리*




>>>>> GPR setting

R7: key base addr

R20: hex base addr

R9: Hex display digit

R11: display limit (9)

Sp: stack pointer (초기값 0x20000)

R2: return value of SEG7_CODE

R6: polling key press

R5: polling key release

R4: detect which key pressed

R10: OFF detect

다음 프로그램은 polled IO를 통해 MMIO를 control하는 프로그램이다. 이 프로그램에서는 KEY의 press-release를 polling하여 KEY가 눌렸는지를 확인하고 그에 따라 HEX에 표시되는 숫자를 늘리거나 줄이고, 초기화하거나 OFF하는 동작을 수행하는 프로그램이다.

<GPR setting>

이 프로그램에서 사용하고자 하는 MMIO인 KEY와 HEX의 base address를 각각 r7, r20에 저장한다. 이 reg에 저장된 주소값을 바탕으로 각 MMIO의 parallel port의 register 주소를 찾을 수 있다. MMIO는 parallel port의 reg에 값을 저장하거나 읽어오는 방식으로 control할 수 있게 된다. R9을 통해서 HEX에 display하고자 하는 숫자의 값을 저장하였고, r11을 통해서 프로그램이 한 자리 수에 대해서만 실행될 수 있도록 하였다. 이 프로그램의 OFF subroutine을 실행할 때 RESET subroutine call을 하게 되므로, sp을 사용하여 return address를 저장하는 것을 통해 ret instruction을 올바르게 수행할 수 있도록 하였다.

<KEY_PRESS_WAIT, KEY_RELEASE_WAIT>

이 프로그램에서 polled IO를 구현하는 부분이다. Polled IO란 한 device의 상태를 계속 sampling하여 변화되는 순간을 찾아내 IO를 control하는 하나의 전략이다. 이 프로그램의 경우에는 KEY의 상태를 계속 sampling하게 된다. Subroutine의 이

름에서 유추할 수 있듯이, KEY의 press와 release를 연속적으로 관찰하여 두 subroutine의 결과가 만족된다면 KEY device가 press된 event가 발생했다고 판단하게 되는 subroutine이다. MMIO는 parallel port를 통해 control되므로 KEY의 상태를 sampling하기 위해 KEY의 data register에 저장된 값(4bit)을 stw하며 0이 아니게 되는 순간을 찾는다. Release가 되어야 KEY를 한번 press했다고 판단할 수 있으므로 data register의 값이 다시 0이 되는지도 확인한 후 KEY가 press되었다고 판단하게 된다.

<KEY_DETECT>

KEY가 press된 것을 확인한 뒤 어떤 KEY가 눌렸는지도 확인할 필요가 있다. KEY의 data register값을 순차적으로 shift하고 0001과 andi 연산을 통해 나온 값이 0이 되지 않을 때를 찾아 몇 번째 KEY가 press 되었는 지를 알 수 있다. Press된 KEY의 index에 따라 정해진 subroutine을 실행하게 된다.

<RESET>

HEX에 display될 값을 0으로 초기화한다.

<INCREMENT>

HEX에 display될 값이 9보다 작다면 1을 더한 뒤 return 한다. 9보다 작지 않다면 그대로 return 하여 9가 display되게 한다.

<DECREMENT>

HEX에 display될 값이 0이라면 그래도 return하고, 그렇지 않다면 1을 뺀 뒤 return 한다.

<OFF>

HEX에 display될 값을 0으로 초기화 하고, HEX에 아무것도 display하지 않는다. 즉 모든 7segment의 값을 0으로 한다. 이 때 RESET subroutine과 값을 0으로 초기화하는 부분이 같으므로 RESET subroutine call을 통해 구현하게 된다. OFF를 올바르게 ret하기 위해서 sp를 사용하여 return address를 저장하였다.

<SEG7_CODE>

Lab1에서 사용하였던 코드를 OFF subroutine을 구분하는 문장을 추가한 후 그대로 사용하였다. Display 하고자 하는 값에 해당되는 BCD값을 HEX base addr에 stw한다. OFF subroutine이 실행될 시에는 <SEG7_OFF>를 통해 0값을 HEX base addr에 stw한다.

*구현 코드 설명*

**** -io instructions (stwio, ldwio) : cache bypass instruction이다. MMIO를 control할 때는 new data를 즉각적으로 access 해야하기 때문에 cache bypass instrcution을 사용하게 된다.

```
.text
.global _start

_start:   movia r7, 0xFF200050      ## KEY0 base addr
```

```
        movia r20, 0xFF200020     ## HEX3_HEX0 base addr
        mov r9, r0                ## HEX display digit
        movi r11, 9               ## display limit (execute for single digit)
        movia sp, 0x20000         ## use stack to save return address

MAIN:   call SEG7_CODE
        stwio r2, (r20)

KEY_PRESS_WAIT:     ldwio r6, 0(r7)                ## polling KEY press
                    beq r6, r0, KEY_PRESS_WAIT   ## polling KEY press

KEY_RELEASE_WAIT:   ldwio r5, 0(r7)                  ## polling KEY release
                    bne r5, r0, KEY_RELEASE_WAIT   ## polling KEY release
                    br KEY_DETECT

KEY_DETECT:
                andi r4, r6, 0x1                ## if KEY press, KEY value=0
                bne   r4, r0, RESET_CALL          ## detect which KEY pressed
                srli r6, r6, 0x1
                andi r4, r6, 0x1
                bne   r4, r0, INCREMENT_CALL
                srli r6, r6, 0x1
                andi r4, r6, 0x1
                bne   r4, r0, DECREMENT_CALL
                srli r6, r6, 0x1
                andi r4, r6, 0x1
                bne   r4, r0, OFF_CALL



INCREMENT_CALL: call INCREMENT
                br MAIN
DECREMENT_CALL: call DECREMENT
                br MAIN
RESET_CALL:     call RESET
                br MAIN
OFF_CALL:       call OFF
                br MAIN



INCREMENT:   bge     r9, r11, MAIN          ## if display digit==9, return.
             addi    r9, r9, 1              ## display digit ++
             ret
DECREMENT:   beq     r9, r0, MAIN          ## if display digit==0, return
             subi    r9, r9, 1             ## display digit --
```

```
                ret
RESET:      mov     r9, r0                  ## display digit = 0
            ret
OFF:        subi    sp, sp, 4           ## set space to save return address
            stw     ra, 0(sp)           ## save return address at stack

            call    RESET               ## reset display digit to 0
            movi    r10, 1              ## OFF = TRUE

            ldw     ra, 0(sp)           ## restore return address
            addi    sp, sp, 4           ## restore stack pointer

            ret

SEG7_CODE:   bne r10, r0, SEG7_OFF     ##   (OFF == TRUE) ?
             movia r15, BIT_CODES      ##   starting address of bit codes
             add r15, r15, r9          ##   index into the bit codes
             ldb r2, (r15)             ##   read the bit code needed for our digit
             ret
SEG7_OFF:   ldw r2, HEX_OFF(r0)         ## return 0 (BCD)
            mov r10, r0                 ## OFF detector reset
            ret

BIT_CODES: .byte 0b00111111, 0b00000110, 0b01011011, 0b01001111, 0b01100110
           .byte 0b01101101, 0b01111101, 0b00000111, 0b01111111, 0b01100111
          .skip   2      # pad with 2 bytes to maintain word alignment
HEX_OFF:   .word 0b00000000
          .end
```

*결과 및 토의*

KEY 0-3을 눌렀을 때, KEY data register의 값을 r6를 통해 확인한다. 각 key의 번호에 따라 4비트중 한자리의 값이 1이 될 것이다. Debugger를 통해 확인한다면 16진수로 표현되므로 1,2,4,8중 한 값이 r6에 저장될 것이다.

KEY를 떼는 순간 KEY data register의 값이 0되므로 r5를 통해 값이 0인지를 확인할 수 있다.

보드에서 실행하면 KEY release를 한 후 빠른 속도로 subroutine 실행 후 ret하기 때문에 debugger에서 r5, r6의 값은 같은 값으로 확인된다.

HEX에 display되는 숫자와 r9에 저장되어 있는 숫자가 같은 수인지를 확인하여 결과값이 맞게 나오는지 확인할 수 있다. 또한 display하고자 하는 수가 9보다 클 때 KEY1을 누르면 9의 값이 그대로 display되고, 0보다 작을 때 KEY2를 누르면 0이 그대로 display되는 것을 확인하였다.

| | |
|---|---|
| pc | 0x0000002C |
| zero | 0x00000000 |
| r1 | 0x00000000 |
| r2 | 0x0000003F |
| r3 | 0x00000000 |
| r4 | 0x00000001 |
| r5 | 0x00000000 |
| r6 | 0x00000000 |
| r7 | 0xFF200050 |
| r8 | 0x00000000 |
| r9 | 0x00000000 |
| r10 | 0x00000000 |
| r11 | 0x00000009 |
| r12 | 0x00000000 |
| r13 | 0x00000000 |
| r14 | 0x00000000 |
| r15 | 0x000000E8 |
| r16 | 0x00000000 |
| r17 | 0x00000000 |
| r18 | 0x00000000 |
| r19 | 0x00000000 |
| r20 | 0xFF200020 |
| r21 | 0x00000000 |
| r22 | 0x00000000 |
| r23 | 0x00000000 |
| et | 0x00000000 |
| bt | 0xFFFFFFFF |
| gp | 0x00000000 |
| sp | 0x00020000 |
| fp | 0x00000000 |
| ea | 0x00000000 |
| ba | 0xFFFFFFFF |
| ra | 0x00000024 |
| status | 0x00000000 |

>>> 처음상태

| | |
|---|---|
| pc | 0x00000034 |
| zero | 0x00000000 |
| r1 | 0x00000000 |
| r2 | 0x00000006 |
| r3 | 0x00000000 |
| r4 | 0x00000001 |
| r5 | 0x00000002 |
| r6 | 0x00000002 |
| r7 | 0xFF200050 |
| r8 | 0x00000000 |
| r9 | 0x00000001 |
| r10 | 0x00000000 |
| r11 | 0x00000009 |
| r12 | 0x00000000 |
| r13 | 0x00000000 |
| r14 | 0x00000000 |
| r15 | 0x000000E9 |
| r16 | 0x00000000 |
| r17 | 0x00000000 |
| r18 | 0x00000000 |
| r19 | 0x00000000 |
| r20 | 0xFF200020 |
| r21 | 0x00000000 |
| r22 | 0x00000000 |
| r23 | 0x00000000 |
| et | 0x00000000 |
| bt | 0xFFFFFFFF |
| gp | 0x00000000 |
| sp | 0x00020000 |
| fp | 0x00000000 |
| ea | 0x00000000 |
| ba | 0xFFFFFFFF |
| ra | 0x00000024 |

>>> key1 누른 상태

>>> key를 release 하면서 INCREMENT 실행

| | |
|---|---|
| pc | 0x00000028 |
| zero | 0x00000000 |
| r1 | 0x00000000 |
| r2 | 0x0000005B |
| r3 | 0x00000000 |
| r4 | 0x00000001 |
| r5 | 0x00000000 |
| r6 | 0x00000000 |
| r7 | 0xFF200050 |
| r8 | 0x00000000 |
| r9 | 0x00000002 |
| r10 | 0x00000000 |
| r11 | 0x00000009 |
| r12 | 0x00000000 |
| r13 | 0x00000000 |
| r14 | 0x00000000 |
| r15 | 0x000000EA |
| r16 | 0x00000000 |
| r17 | 0x00000000 |
| r18 | 0x00000000 |
| r19 | 0x00000000 |
| r20 | 0xFF200020 |
| r21 | 0x00000000 |
| r22 | 0x00000000 |
| r23 | 0x00000000 |
| et | 0x00000000 |
| bt | 0xFFFFFFFF |
| gp | 0x00000000 |
| sp | 0x00020000 |
| fp | 0x00000000 |
| ea | 0x00000000 |
| ba | 0xFFFFFFFF |
| ra | 0x00000024 |

>>>key1 뗀 상태

| | |
|---|---|
| pc | 0x00000030 |
| zero | 0x00000000 |
| r1 | 0x00000000 |
| r2 | 0x0000005B |
| r3 | 0x00000000 |
| r4 | 0x00000001 |
| r5 | 0x00000008 |
| r6 | 0x00000008 |
| r7 | 0xFF200050 |
| r8 | 0x00000000 |
| r9 | 0x00000002 |
| r10 | 0x00000000 |
| r11 | 0x00000009 |
| r12 | 0x00000000 |
| r13 | 0x00000000 |
| r14 | 0x00000000 |
| r15 | 0x000000EA |
| r16 | 0x00000000 |
| r17 | 0x00000000 |
| r18 | 0x00000000 |
| r19 | 0x00000000 |
| r20 | 0xFF200020 |
| r21 | 0x00000000 |
| r22 | 0x00000000 |
| r23 | 0x00000000 |
| et | 0x00000000 |
| bt | 0xFFFFFFFF |
| gp | 0x00000000 |
| sp | 0x00020000 |
| fp | 0x00000000 |
| ea | 0x00000000 |
| ba | 0xFFFFFFFF |
| ra | 0x00000024 |

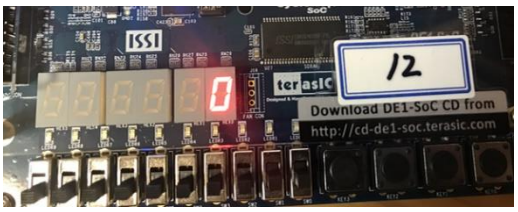
>>> key3 누른 상태

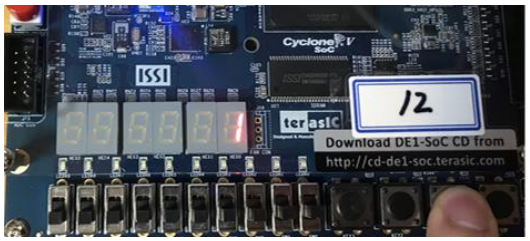
>>>key를 release하면서 OFF 실행

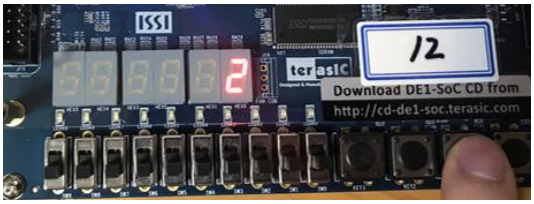
>>>key를 release한 후 subroutine으로 가므로 sp 변화는 확인하지 못함. (이미 restore 된 상태이다.)

| | |
|---|---|
| pc | 0x0000002C |
| zero | 0x00000000 |
| r1 | 0x00000000 |
| r2 | 0x00000000 |
| r3 | 0x00000000 |
| r4 | 0x00000001 |
| r5 | 0x00000000 |
| r6 | 0x00000000 |
| r7 | 0xFF200050 |
| r8 | 0x00000000 |
| r9 | 0x00000000 |
| r10 | 0x00000000 |
| r11 | 0x00000009 |
| r12 | 0x00000000 |
| r13 | 0x00000000 |
| r14 | 0x00000000 |
| r15 | 0x000000EA |
| r16 | 0x00000000 |
| r17 | 0x00000000 |
| r18 | 0x00000000 |
| r19 | 0x00000000 |
| r20 | 0xFF200020 |
| r21 | 0x00000000 |
| r22 | 0x00000000 |
| r23 | 0x00000000 |
| et | 0x00000000 |
| bt | 0xFFFFFFFF |
| gp | 0x00000000 |
| sp | 0x00020000 |
| fp | 0x00000000 |
| ea | 0x00000000 |
| ba | 0xFFFFFFFF |
| ra | 0x00000024 |

>>>key3 뗀 상태



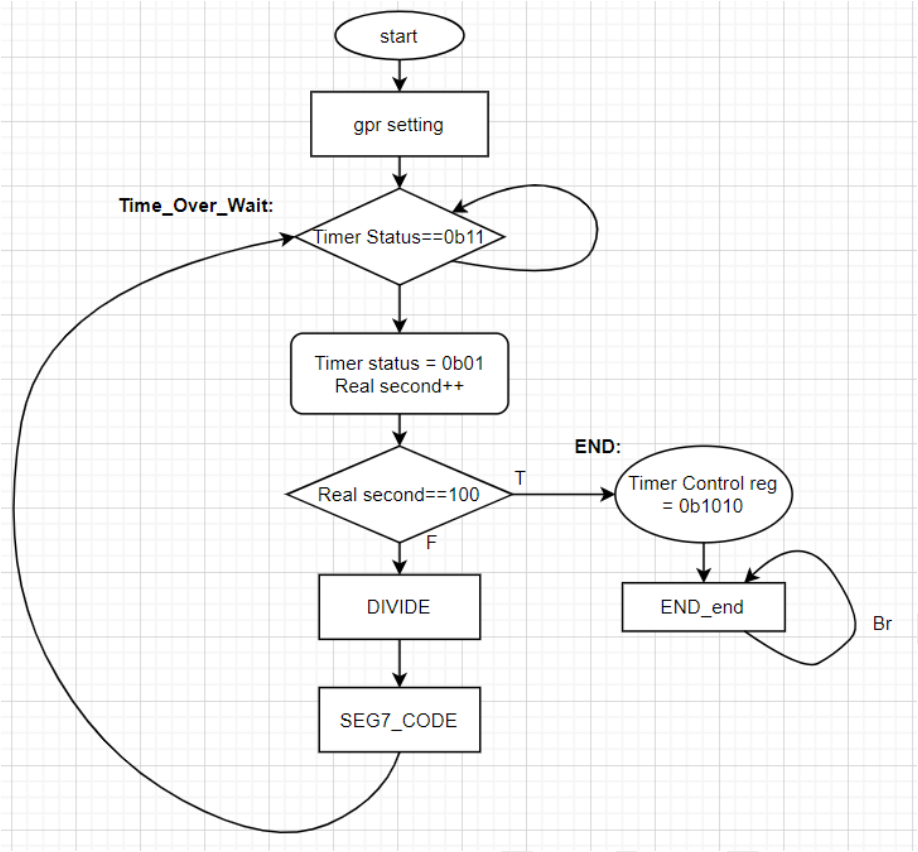




# 전자 HW 설계 – 실습 보고서

## ✓ Part II

*동작 원리*




>>>>> GPR setting

R18: Timer base addr

R23: 100 (Timer limit)

R22: HEX base addr

R20: period value

R9: real second

R10: status reg initial value

R21: control reg initial value

R11, r12: polling Timer status

DE1-soC computer platform 내부에 존재하는 Interval timer를 통해 실제시간과 똑같은 타이머를 만드는 프로그램이다. 0부터 99까지 실제 시간과 같은 속도로 시간을 세고 99가 되면 멈추게 된다.

<GPR setting>

R18을 통해 interval timer의 base address를 저장한다. 이 주소를 통해 interval timer의 parallel port의 register에 접근할 수 있다. R22에는 HEX의 base address를 저장하였다. R23에 타이머의 limit값을 저장하고 저장된 값에 도달하면 더 이상 시간을 세지 않도록 하였다. R20을 통해 interval timer의 period를 저장하게 하고, r9을 증가함으로써 증가된 시간을 display할 수 있게 한다. 이 때 DE1-soC 내부의 clock은 100MHz이므로 실제시간과 같은 1초를 세기 위해서는 period의 값을 10^8로 설정해야 했다. R10은 status register의 initial 값(0b10)을 저장한다. R21은 control register의 initial 값(0b0110)을 저장한다. R11, r12를 통해서 Timer의 status register를 polling한다. 즉 reg의 TO값이 1이 되는지를 확인한다.

< Time_Over_Wait>

Status register의 값을 polling하는 subroutine이다. TO값이 1이되는지를 확인하기 위해 즉 status reg value가 0b11이 되는지를 확인하기 위해 status reg+1한 값과 status reg의 값을 and연산 취하여 0이되는지를 지속적으로 확인하는 subroutine이다. 만약 TO인 것을 확인하였다면 display 할 value에 1을 더하고 status reg의 값을 다시 초기화 하는 과정을 거친다. 다음 시간도 세기 위해서는 reg값을 초기화해 줄 필요가 있다. 또한 display할 value가 100이 되었다면 display하지 않고 END하도록 한다.

<END>

Timer가 99까지 세었다면 멈추게 하는 subroutine이다. 이 때 display하는 값을 일정하게 하는 것 뿐만 아니라 내부 timer의 동작도 멈추게 하기 위해 Control register의 값도 0b1010으로 바꿔준다. (STOP 비트를 1로 만들고 START 비트를 0으로 한다. ) 이후 무한루프를 돌게 된다.

*구현 코드 설명*

*** Movui : Timer parallel port의 register는 16bit로 구성되어 있기 때문에 periodl, periodh를 설정할 때 unsigned value로 저장하지 않는다면 overflow가 발생하여 음수의 값으로 인식하게 된다. 10^8의 값을 저장하고 싶다면 unsigned instruction을 사용해야 한다.

```
.text
.global _start

_start:     movia r18, 0xff202000          ## Timer_base_addr
           movi   r23, 100
            movia r22, 0xff200020          ## HEX_base_addr
            movui r20, 0xe100              ## period_l setting
            stwio r20, 8(r18)
            movui r20, 0x5f5               ## period_h setting
            stwio r20, 12(r18)
            mov    r9, r0                  ## timer digit
            ldw    r10, INITIAL(r0)
            stwio r10, (r18)               ## initial status
            ldw    r21, CONTROL(r0)
            stwio r21, 4(r18)              ## timer control

Time_Over_Wait:  ldwio   r11, (r18)              ## status reg read
                 addi r12, r11, 1
                 and   r12, r12, r11
                 bne   r12, r0, Time_Over_Wait   ## polling TO
                 stwio r10, (r18)
                 addi r9, r9, 1                  ## display value ++
                 beq   r9, r23, END              ## at 99 sec, END


DISPLAY:          call DIVIDE          # ones digit will be in r2; tens digit in r3
                  mov r4, r2           # pass ones digit to SEG7_CODE
                  call SEG7_CODE
                  mov r14, r2          # save bit code
                  mov r4, r3           # retrieve tens digit, pass to SEG7_CODE
                  call SEG7_CODE
                  slli r2, r2, 8
```

```
                or r14, r14, r2          # bit code for tens|bit code for ones
                stwio r14, (r22)         ## stwio at HEX base addr
                br Time_Over_Wait     ## polling again



INITIAL: .word 0b10
CONTROL: .word 0b0110


DIVIDE: mov      r2, r9       # r5로 계산
         movi    r5, 10      # divider
         movi    r3, 0       # 몫

CONT:   blt r2, r5, DIV_END   # divider보다 작을 때 까지 뺄셈 수행
        sub r2, r2, r5        # 나머지 계산
        addi r3, r3, 1        # 몫 계산
        br CONT

DIV_END: ret

SEG7_CODE: movia r15, BIT_CODES # starting address of bit codes
           add r15, r15, r4 # index into the bit codes
           ldb r2, (r15) # read the bit code needed for our digit
           ret

BIT_CODES: .byte 0b00111111, 0b00000110, 0b01011011, 0b01001111, 0b01100110
           .byte 0b01101101, 0b01111101, 0b00000111, 0b01111111, 0b01100111
                    .skip   2   # pad with 2 bytes to maintain word alignment

END:        movi r24, 0b1010        ## timer stop
            stbio r24, 4(r18)       ## stbio at KEY control register address

END_end:   br END_end               ## program END
           .end
```

*결과 및 토의*

Timer가 시작할 때 r11, r21의 값을 통해서 status reg와 control reg의 값이 0b10, 0b0110으로 설정되어 있는 것을 확인할 수 있다. 또한 시간이 흘러가면서 HEX에 display되는 값과 r9에 저장되어 있는 값을 비교하면서 올바르게 동작하고 있는 것을 확인할 수 있다. Timer가 끝나게 되면 r11, r24를 통해서 status reg와 control reg가 0b11, 0b1010으로 설정되어 멈추게 된 것을 확인할 수 있다.

Registers

| Reg | Value |
|---|---|
| pc | 0x0000003C |
| zero | 0x00000000 |
| r1 | 0x00000000 |
| r2 | 0x00003F00 |
| r3 | 0x00000000 |
| r4 | 0x00000000 |
| r5 | 0x0000000A |
| r6 | 0x00000000 |
| r7 | 0x00000000 |
| r8 | 0x00000000 |
| r9 | 0x00000004 |
| r10 | 0x00000002 |
| r11 | 0x00000002 |
| r12 | 0x00000002 |
| r13 | 0x00000000 |
| r14 | 0x00003F66 |
| r15 | 0x000000B8 |
| r16 | 0x00000000 |
| r17 | 0x00000000 |
| r18 | 0xFF202000 |
| r19 | 0x00000000 |
| r20 | 0x000005F5 |
| r21 | 0x00000006 |
| r22 | 0xFF200020 |
| r23 | 0x00000064 |
| et | 0x00000000 |
| bt | 0xFFFFFFFF |
| gp | 0x00000000 |
| sp | 0x00000000 |
| fp | 0x00000000 |
| ea | 0x00000000 |
| ba | 0xFFFFFFFF |
| ra | 0x0000006C |
| status | 0x00000000 |
| estatus | 0x00000000 |
| bstatus | 0xFFFFFFFF |
| ienable | 0x00000000 |
| ipending | 0x00000000 |
| cpuid | 0x00000000 |

>>> 99까지 세고 종료

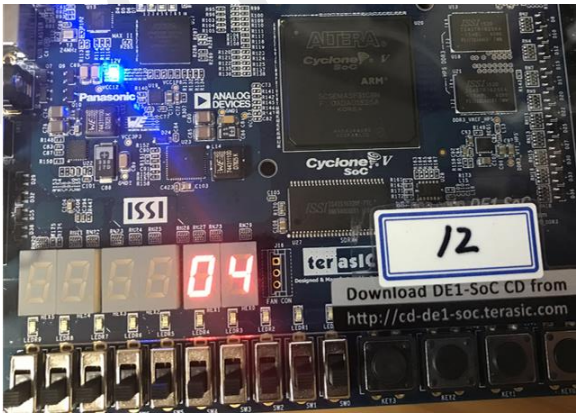
>>> status 0b11로 종료

>>>control 0b1010로 종료

Registers

| Reg | Value |
|---|---|
| pc | 0x000000CC |
| zero | 0x00000000 |
| r1 | 0x00000000 |
| r2 | 0x00006700 |
| r3 | 0x00000009 |
| r4 | 0x00000009 |
| r5 | 0x0000000A |
| r6 | 0x00000000 |
| r7 | 0x00000000 |
| r8 | 0x00000000 |
| r9 | 0x00000064 |
| r10 | 0x00000002 |
| r11 | 0x00000003 |
| r12 | 0x00000000 |
| r13 | 0x00000000 |
| r14 | 0x00006767 |
| r15 | 0x000000C1 |
| r16 | 0x00000000 |
| r17 | 0x00000000 |
| r18 | 0xFF202000 |
| r19 | 0x00000000 |
| r20 | 0x000005F5 |
| r21 | 0x00000006 |
| r22 | 0xFF200020 |
| r23 | 0x00000064 |
| et | 0x0000000A |
| bt | 0xFFFFFFFF |
| gp | 0x00000000 |
| sp | 0x00000000 |
| fp | 0x00000000 |
| ea | 0x00000000 |
| ba | 0xFFFFFFFF |
| ra | 0x0000006C |
| status | 0x00000000 |
| estatus | 0x00000000 |
| bstatus | 0xFFFFFFFF |
| ienable | 0x00000000 |
| ipending | 0x00000000 |
| cpuid | 0x00000000 |

>>> 타이머가 4초 일 때




>>> 타이머 종료 시