

전자 HW 설계 – 실습 보고서

이름: 박관영 (2016124099)

✓ Part I

동작 원리

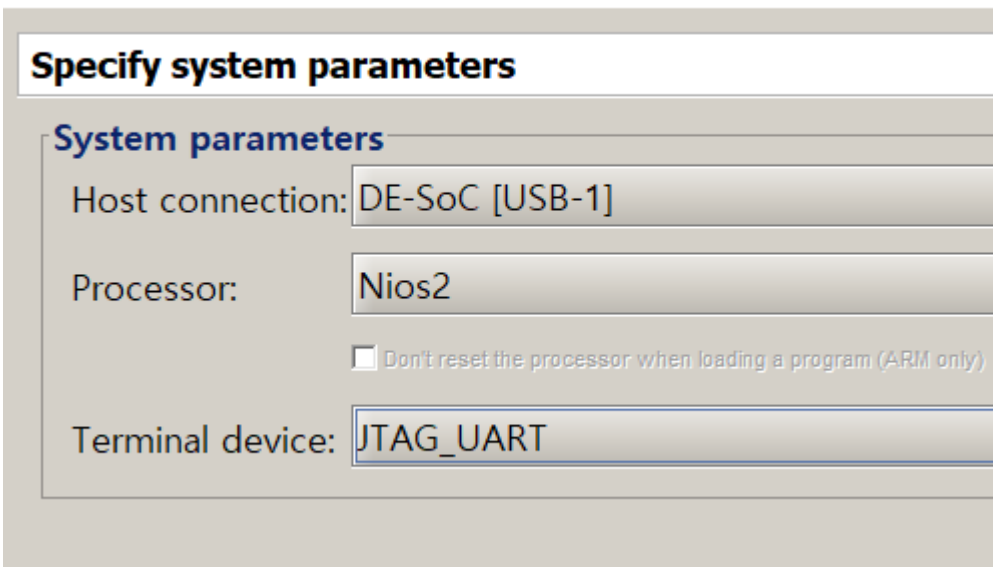
배열을 선언하고, 그 최대값을 찾아 출력하는 프로그램이다. 기본적인 c코드를 보드위에 올려보는 실험이다.

구현 코드 설명

```
#include <stdio.h>
void main() {
    int LIST[8] = { 7,4,5,3,6,1,8,2 };
    int i;
    int k = LIST[0];          // LIST의 원소 개수 detect
    int max = 0;              //최댓값 저장할 변수 선언
    for (i = 1; i < k; i++) {  // LIST의 원소 개수만큼 반복문 실행
        if (max < LIST[i]) max = LIST[i];    // max에 저장된 값보다 큰 원소의 값을 찾으면 max값 update
    }
    printf("Max value is %d", max);        // 마지막에 저장되어 있는 max 값 출력
}
```

결과 및 토의

New Project Wizard



Terminal

JTAG UART link established using cable "DE-SoC [USB-1]", device 2, instance 0x00
Max value is 8

Printf 함수의 출력은 monitor program을 만들 때 설정한 terminal인 JTAG_UART로 나오게 되는 것을 확인할 수 있다.

전자 HW 설계 – 실습 보고서

✓ Part II

동작 원리

Part1과 같은 프로그램이지만 출력을 LEDR을 통해 확인하는 실험이다. memory mapped register를 제어할 때는 volatile 타입의 변수를 통해 컴파일과정에서 자동적으로 최적화를 하는 것을 막는 것이 필요하다. memory mapped register의 경우 프로그램이 진행되는 과정에서 하드웨어에 의해 그 값이 바뀔 수 있기 때문에 volatile 형태로 선언하여 컴파일 과정에서의 최적화를 못하게 해야 한다.

구현 코드 설명

```
#include <stdio.h>

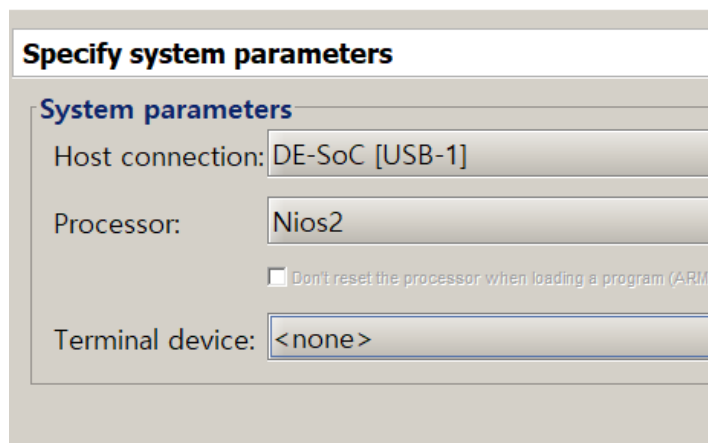
#define LEDR_BASE 0xFF200000          // LEDR base addr define
static volatile int *LEDR_ptr = (int*)LEDR_BASE;    // static volatile type 선언

void display_led(int number){          // ledr memory mapped reg에 value access
    *LEDR_ptr=number;
}

void main() {
    int LIST[8] = { 7,4,5,3,6,1,8,2 };
    int i;
    int k = LIST[0];                    // LIST의 원소 개수 detect
    int max = 0;                        //최댓값 저장할 변수 선언
    for (i = 1; i < k; i++) {           // LIST의 원소 개수만큼 반복문 실행
        if (max < LIST[i]) max = LIST[i];    // max에 저장된 값보다 큰 원소의 값을 찾으면 max값 update
    }
    display_led(max);                   // max value를 인자로 전달하여 LEDR에 출력
}
```

결과 및 토의

New Project Wizard



Specify system parameters

System parameters

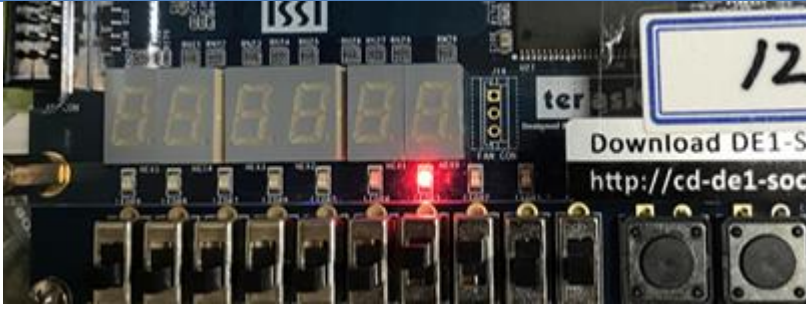
Host connection: DE-SoC [USB-1]

Processor: Nios2

☐ Don't reset the processor when loading a program (ARM)

Terminal device: <none>

전자 HW 설계 – 실습 보고서

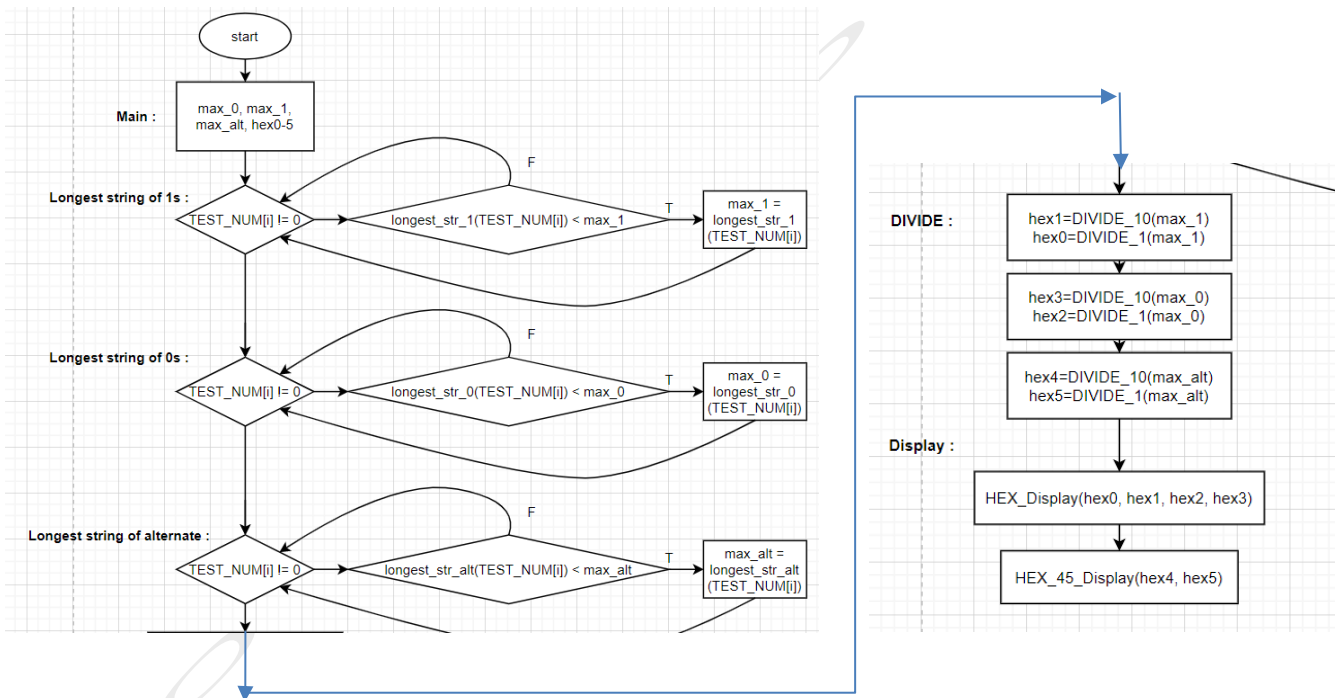


LIST의 최댓값(8)이 LEDR에 나타난 것을 볼 수 있다. Part1과 같은 결과이지만 출력방식이 다르다. Part2의 경우 printf함수를 사용하지 않기 때문에 terminal device를 none으로 설정하여도 올바르게 출력이 되는 것을 볼 수 있다.

✓ Part III

동작 원리

Lab2에서 구현하였던 프로그램과 동일하게 longest string of 1s, 0s, alternate를 찾아내는 프로그램이다. 그 값을 hex에 출력하도록 한다. 구현하고자 하는 프로그램에서 사용할 기능들을 함수로 미리 구현을 한 후 main함수에서 순차적으로 call하여 프로그램을 완성하였다.



구현 코드 설명

<global variables setting>

사용하고자 하는 memory mapped reg의 주소를 매크로로 정의한다. 또한 이 주소에 접근하여 reg에 값을 저장하기 위해 static volatile int 타입의 포인터 변수를 선언하여 mmio를 제어한다. 이 후 함수에서 사용될 숫자들의 집합인 TEST_NUM은 shift 연산을 통해 동작이 구현되므로 unsigned int 타입으로 선언하여 0으로 채워지게 한다(signed로 선언하게 되면 shift 된 자리가 1로 채워지게 되어 함수 내부에서 무한루프를 돌게 되는 오류가 발생한다). HEX에 display될 BCD 값도 전역변수로 미리 선언하여 후에 자유롭게 사용한다.

전자 HW 설계 – 실습 보고서

<HEX_display, HEX_45_display>

Static volatile 타입으로 선언된 memory mapped register에 display하고자 하는 수의 BCD값을 저장하는 함수이다. HEX3-0과 HEX 5-4가 한 reg로 되어있으므로 각 reg에 대한 함수를 사용하여 display하였다. 이 때 shift 연산을 통해 입력된 수의 hex 번호에 따라 shift를 해주고 or 연산을 통해 각 hex에 표현될 BCD값이 하나의 32bit값으로 나타나게 하였다. 이를 HEX3-0, HEX5-4에 대해 수행하고 각 reg에 store하여 display함수를 구현하였다.

<longest_str_1>

Lab2에서 구현한 assembly code에서 사용한 방법과 동일한 방법으로 구현하였다. TEST_NUM의 원소를 argument로 전달받아 shift 연산과 &연산을 전달된 값이 0이 될 때까지 수행한다. 연산이 수행된 횟수가 곧 가장 긴 연속된 1의 길이라고 할 수 있다. while문을 사용하여 구현할 수 있었다.

<longest_str_0>

argument로 전달받은 TEST_NUM의 원소를 bit 반전시킨 후 longest_str_1을 call한다. 이 때 return되는 값이 argument의 가장 긴 연속된 0의 길이라고 할 수 있다.

<longest_str_alt>

0,1이 연속되는 길이를 찾아내는 것이므로 비교할 immediate인 cmp를 0xaaaaaaaa로 초기화 한다(이진수로 101010...이 반복되는 32bit immediate). argument로 전달받은 TEST_NUM의 원소와 xor연산을 통해 나온 결과값을 longest_str_0과 longest_str_1에 argument로 전달하고 두 return 값 중 큰 값을 longest_str_alt의 return값으로 정한다. 1010...이나 0101...모두 alternative string이기 때문에 두 값을 비교하는 과정이 필요하다.

<DIVIDE_10>

함수의 결과값을 HEX에 전달할 때 자릿수를 나눠서 10진수로 전달하기 위해 표현할 값의 10의 자리를 구하는 함수이다.

<DIVIDE_1>

함수의 결과값을 HEX에 전달할 때 자릿수를 나눠서 10진수로 전달하기 위해 표현할 값의 1의 자리를 구하는 함수이다.

<main>

위에서 선언한 함수를 사용하여 실제 TEST_NUM의 원소들의 longest string을 구한다. 원소가 0이 되면 끝난 것이므로 0이 아닐 때까지 반복문을 통해 longest_str_1, longest_str_0, longest_str_alt를 실행한다. 전체 원소 중 가장 큰 longest string값을 각각 max_0, max_1, max_alt에 저장하고 DIVIDE, Display함수를 거쳐 HEX device에 출력하게 된다.

```
#include <stdio.h>

#define HEX_BASE 0xFF200020
#define HEX_BASE_45 0xFF200030

unsigned int TEST_NUM[] = { 0x0000e000, 0x3fabedef, 0x00000001, 0x00000002,
                           0x75a5a5a5, 0x01ffc000, 0x03ffc000, 0x55555555,
```

전자 HW 설계 – 실습 보고서

```
0x77777777,0x08888888,0x00000000 );
/*unsigned int로 선언하여 shift될 때 0으로 채워지게 한다.*/

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

static volatile int* HEX_ptr = (int*)HEX_BASE;
static volatile int* HEX_ptr_45=(int*)HEX_BASE_45;
/* static volatile로 mmio 제어 */

void HEX_display(int number_0, int number_1, int number_2, int number_3) {
    *HEX_ptr=BCD[number_0]|BCD[number_1]<<8|BCD[number_2]<<16|BCD[number_3]<<24;
}
/* 네 개의 숫자를 받아서 bit shift를 통해 HEX 0-3에 display되게 한다.*/

void HEX_45_display(int number_4, int number_5){
    *HEX_ptr_45=BCD[number_4]|BCD[number_5]<<8;
}
/* 두 개의 숫자를 받아서 bit shift를 통해 HEX 4-5에 display되게 한다.*/

int longest_str_1(unsigned int number) {
    int shifted;
    int string = 0;
    if (number == 0) return 0;
    while (number != 0) {
        shifted = number >> 1;
        number = shifted & number;
        string++;
    }
    return string;
}
/* shift right와 &연산을 통해 가장 긴 연속된 1의 길이를 알아낸다. */

int longest_str_0(unsigned int number) {
    number = ~number;
    int string;
    string = longest_str_1(number);
    return string;
}
/* 비트 반전된 숫자를 longest_str_1함수에 넣어 가장 긴 연속된 0의 길이를 알아낸다. */

int longest_str_alt(unsigned int number){
    unsigned int cmp=0xaaaaaaaa;
    int string_1, string_0;
```

전자 HW 설계 – 실습 보고서

```
int string;
int value=number^cmp;
string_0=longest_str_0(value);
string_1=longest_str_1(value);
string=(string_0>=string_1)?string_0:string_1;
return string;
}
/* 32비트 모두 1, 0이 번갈아가며 있는 숫자인 0xaaaaaaaa와 argument를 xor취한 후 longest_str_0과 longest_str_1중 더
긴 값을 longest_str_alt로 정한다. */

int DIVIDE_1(unsigned int number) {
    int digit_1;
    digit_1 = number % 10;
    return digit_1;
}
/* display 하고자 하는 값의 1의 자리 수 */
int DIVIDE_10(int number) {
    int digit_10;
    digit_10 = number / 10;
    return digit_10;
}
/* display 하고자 하는 값의 10의 자리 수 */

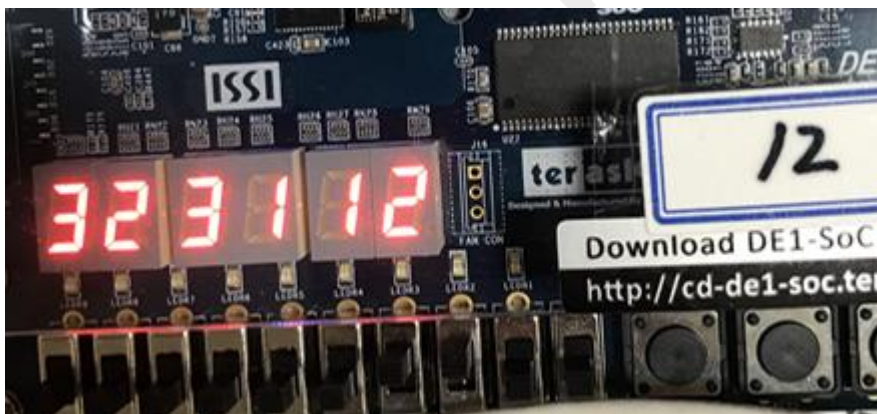
int main() {
    int i=0;
    int max_1 = 0;
    int max_0 = 0;
    int max_alt=0;
    int hex0, hex1, hex2, hex3, hex4, hex5;
    int str_1, str_0, str_alt;
    while(TEST_NUM[i]!=0) {                                // 0을 만날 때까지 TEST_NUM의 값 비교
        str_1 = longest_str_1(TEST_NUM[i]);
        if (max_1 < str_1) {
            max_1 = str_1;
        }
        i++;
    }
    i=0;
    while(TEST_NUM[i]!=0) {
        str_0 = longest_str_0(TEST_NUM[i]);

        if (max_0 < str_0) {
            max_0 = str_0;
        }
        i++;
    }
}
```

전자 HW 설계 – 실습 보고서

```
}  
i=0;  
while(TEST_NUM[i]!=0) {  
    str_alt = longest_str_alt(TEST_NUM[i]);  
  
    if (max_alt < str_alt) {  
        max_alt = str_alt;  
    }  
    i++;  
}  
  
hex5=DIVIDE_10(max_alt);  
hex4=DIVIDE_1(max_alt);  
hex3 = DIVIDE_10(max_0);  
hex2 = DIVIDE_1(max_0);  
hex1 = DIVIDE_10(max_1);  
hex0 = DIVIDE_1(max_1);  
/* display 하고자 값을 10의 자리와 1의 자리로 나누어 적절한 hex에 store한다. */  
  
HEX_display(hex0, hex1, hex2, hex3);    // 위에서 구현한 display함수에 인수를 넘겨준다  
HEX_45_display(hex4, hex5);  
  
}
```

결과 및 토의

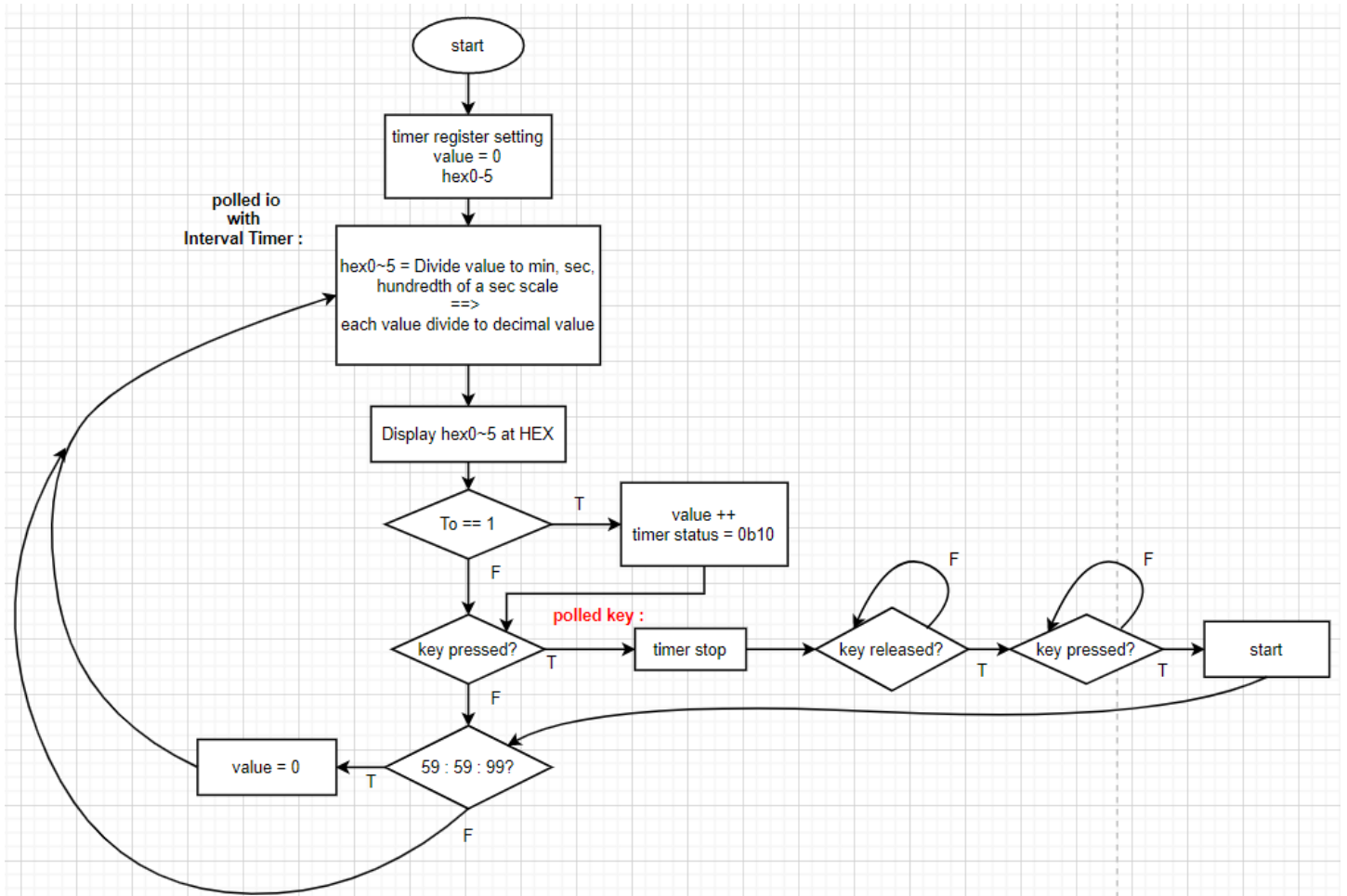


다음과 같이 longest string of alternative, longest string of 0s, longest string of 1s 가 HEX에 순차적으로 출력되는 것을 볼 수 있다.

전자 HW 설계 – 실습 보고서

✓ Part IV

동작 원리



Polled io with the interval timer를 통해 real timer를 구현하는 program이다. 이전 lab과 다르게 1/100 second 단위로 시간을 세게 되므로 period값을 다르게 설정하게 된다. Key를 통해 timer를 start/stop하게 하기 위해 key register의 값도 polling할 필요가 있다. main함수에서 만약 key가 눌리게 된다면 그 시점부터 polled io with the key를 하도록 구현하여 key에 대한 polling도 수행할 수 있도록 하였다. HEX에 display하는 함수와 memory mapped register를 제어하기 위한 volatile type 변수를 선언하는 과정은 part3와 동일하게 수행하였다.

구현 코드 설명

<polled_time_over>

Interval timer의 status register의 To bit를 return 한다. 이 프로그램에서는 0b01과 & 연산을 통해 To bit를 detect 하였다.

<start>

Static volatile type으로 선언된 포인터 변수를 통해 timer control register에 0b0110을 저장한다. 즉 start bit를 on 한다.

<stop>

전자 HW 설계 – 실습 보고서

Static volatile type으로 선언된 포인터 변수를 통해 timer control register에 0b1010을 저장한다. 즉 stop bit를 on 한다.

<DIVIDE_min>

1/100초 단위로 세어진 값을 min 단위로 변환하는 함수이다. 이 프로그램에서는 value/6000을 통해서 값을 결정하였고, 그 값을 /10, %10 연산을 통해 decimal 단위로 표현할 수 있게 하였다.

<DIVIDE_sec>

1/100초 단위로 세어진 값을 sec 단위로 변환하는 함수이다. 이 프로그램에서는 value%6000/100을 통해서 값을 결정하였고, 그 값을 /10, %10 연산을 통해 decimal 단위로 표현할 수 있게 하였다.

<DIVIDE_millisec>

1/100초 단위로 세어진 값을 min, sec 단위로 표현하고 남은 값을 /10, %10을 통해 decimal 단위로 표현하였다.

<main>

While(1)문을 통해 polled io를 구현하였다. 즉, timer의 status register를 polled_time_over 함수의 return 값을 통해 polled io하고 그 return 값이 1이 될 때마다 value를 1씩 더해주며 1/100초 단위의 시간을 세었다. 또한 그 값을 Display, DIVIDE 함수를 통해 HEX에 display하였다. 이러한 과정이 무한루프를 통해 수행되다가 key가 pressed 되는 순간 timer를 stop상태로 만들고 key를 polled io하기 위한 무한 반복문을 수행하게 된다. 여기서는 총 세개의 무한 반복문이 수행되는데 key가 다시 release되는 순간, 다시 key가 press, release되는 순간을 detect하기 위함이다. 각 단계의 조건이 만족되지 않는다면 timer는 계속 stop 상태에 머물게 된다. 만약 모든 반복문을 빠져나오게 된다면 다시 timer는 start 상태로 동작하고 main 함수에서는 timer의 status register를 polled io 하게 된다. 이를 통해 key를 누르면 timer가 멈추고 다시 누르면 timer가 동작하도록 프로그램을 구현할 수 있었다.

```
#include<stdio.h>
#define Timer_base 0xff202000
#define KEY_base 0xff200050
#define HEX_BASE 0xFF200020
#define HEX_BASE_45 0xFF200030

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

static volatile int* Timer_status_ptr = (int*)Timer_base;
static volatile int* Timer_control_ptr = (int*)(Timer_base+4);
static volatile int* Timer_periodl_ptr = (int*)(Timer_base + 8);
static volatile int* Timer_periodh_ptr = (int*)(Timer_base + 12);

static volatile int* KEY_ptr = (int*)(KEY_base);
static volatile int* HEX_ptr = (int*)HEX_BASE;
static volatile int* HEX_ptr_45 = (int*)HEX_BASE_45;
```

전자 HW 설계 – 실습 보고서

```
/* memory mapped register을 제어할 static volatile type의 변수 선언 */

void HEX_display(int number_0, int number_1, int number_2, int number_3) {
    *HEX_ptr = BCD[number_0] | BCD[number_1] << 8 | BCD[number_2] << 16 | BCD[number_3] << 24;
}

void HEX_45_display(int number_4, int number_5) {
    *HEX_ptr_45 = BCD[number_4] | BCD[number_5] << 8;
}

/* part3과 동일한 HEX3-0, HEX5-4 display 함수 */

int polled_time_over() {
    // interval timer의 status register To bit를 polling하기 위한 함수이다.
    int time_over;
    time_over = *Timer_status_ptr & 0b01;
    return (time_over);
}

void start() {
    *Timer_control_ptr = 0b0110; // interval timer control reg에 0110 저장. (시작)
}

void stop() {
    *Timer_control_ptr = 0b1010; // interval timer control reg에 1010 저장. (정지)
}

int DIVIDE_min_5(int number) {
    int value;
    value = number / 6000;
    value = value / 10;
    return value;
}

int DIVIDE_min_4(int number) {
    int value;
    value = number / 6000;
    value = value % 10;
    return value;
}

int DIVIDE_sec_3(int number) {
    int value;
    value = (number % 6000) / 100;
    value = value / 10;
    return value;
}

int DIVIDE_sec_2(int number) {
    int value;
```

전자 HW 설계 – 실습 보고서

```
    value = (number % 6000) / 100;
    value = value % 10;
    return value;
}
int DIVIDE_milise1(int number) {
    int value;
    value = (number % 6000 % 100);
    value = value / 10;
    return value;
}
int DIVIDE_milise0(int number) {
    int value;
    value = (number % 6000 % 100);
    value = value % 10;
    return value;
}
/* hundredths of a second 단위로 시간을 세고 그 value를 각 단위에 맞는 10진수 값을 return 하는 함수들이다. */

int main() {
    int value = 0;
    int hex0, hex1, hex2, hex3, hex4, hex5;
    *Timer_periodl_ptr = 0x4240;
    *Timer_periodh_ptr = 0xf;
    *Timer_status_ptr = 0b10;
    *Timer_control_ptr = 0b0110;
    /* timer register의 초기값을 다음과 같이 설정하여 1/100 초 단위로 timer가 동작하도록 한다. 그 값을 저장할 value 변수와 display 값을 저장할 hex0-5 변수를 선언. */

    while (1) {

        hex0 = DIVIDE_milise0(value);
        hex1 = DIVIDE_milise1(value);
        hex2 = DIVIDE_sec_2(value);
        hex3 = DIVIDE_sec_3(value);
        hex4 = DIVIDE_min_4(value);
        hex5 = DIVIDE_min_5(value);
        HEX_display(hex0, hex1, hex2, hex3);
        HEX_45_display(hex4, hex5);           // 시간 display
        if (polled_time_over() == 1) { value++; *Timer_status_ptr = 0b10; } // timer time over bit polling
        if (*KEY_ptr!=0) {

            stop();
            while(1){
                if(*KEY_ptr==0) { break;}
            }
        }
    }
}
```

전자 HW 설계 – 실습 보고서

```

while(1){
    if(*KEY_ptr!=0) {break;}
}
while(1){
    if(*KEY_ptr==0) {break;}
}
start();
}

/* key가 pressed되면 released 후 다시 pressed 될 때까지 key data polling */
if (value == 359999) { value = 0; }    // 59 : 59 : 99 되면 다시 시작.
}
}

```

결과 및 토의

while문에 의해 timer의 status register가 polling되면서 display 함수가 실행되어 1/100초 단위 타이머가 동작하는 것을 확인할 수 있다. Period를 10^6 으로 설정하였으므로, de1-soc의 기본 clock Hz를 고려해보면 1/100초 단위로 세어질 것이라는 것을 예상할 수 있다. 또한 key가 pressed된 순간, 즉 key parallel port register 중 data reg에 값이 들어오는 순간 key에 대해 polled io를 수행하기 때문에 다시 key가 pressed된 후 released될 때까지 타이머가 계속 stop 상태로 있게 되는 것을 알 수 있다. 그 후 다시 start 함수에 의해 타이머가 정상 작동하는 것을 볼 수 있다. 이후 다시 timer의 status register를 polled io 하게 된다.

main함수의 while문을 수행할 때 위에서 선언한 Display, DIVIDE, start, stop, polled_time_over 등 모든 함수를 call하여 반복문을 계속 수행하면서 timer의 status register를 polled io 하기 때문에 monitor program에서 변환된 assembly code 전체를 반복적으로 수행하는 것을 볼 수 있었다. 그러다 key를 누르는 순간 key를 polling하게 되므로 설정해둔 while(1) 무한 반복문을 연속적으로 수행하는 것을 확인하였다.

<pre> 0x000004E4 00BFC834 orhi r2, zero, 0xFF20 0x000004E8 10801404 addi r2, r2, 0x50 0x000004EC 10800037 ldwio r2, 0(r2) 0x000004F0 103FFC1E bne r2, zero, -0x10 (0x000004E4) } while(1){ if(*KEY_ptr!=0) {break;} 0x000004F4 00BFC834 orhi r2, zero, 0xFF20 0x000004F8 10801404 addi r2, r2, 0x50 0x000004FC 10800037 ldwio r2, 0(r2) 0x00000500 103FFC26 beq r2, zero, -0x10 (0x000004F4) } while(1){ if(*KEY_ptr==0) {break;} 0x00000504 00BFC834 orhi r2, zero, 0xFF20 0x00000508 10801404 addi r2, r2, 0x50 0x0000050C 10800037 ldwio r2, 0(r2) 0x00000510 103FFC1E bne r2, zero, -0x10 (0x00000504) } </pre>	<pre> 0x000004E4 00BFC834 orhi r2, zero, 0xFF20 0x000004E8 10801404 addi r2, r2, 0x50 0x000004EC 10800037 ldwio r2, 0(r2) 0x000004F0 103FFC1E bne r2, zero, -0x10 (0x000004E4) } while(1){ if(*KEY_ptr!=0) {break;} 0x000004F4 00BFC834 orhi r2, zero, 0xFF20 0x000004F8 10801404 addi r2, r2, 0x50 0x000004FC 10800037 ldwio r2, 0(r2) 0x00000500 103FFC26 beq r2, zero, -0x10 (0x000004F4) } while(1){ if(*KEY_ptr==0) {break;} 0x00000504 00BFC834 orhi r2, zero, 0xFF20 0x00000508 10801404 addi r2, r2, 0x50 0x0000050C 10800037 ldwio r2, 0(r2) 0x00000510 103FFC1E bne r2, zero, -0x10 (0x00000504) } </pre>
>>> key를 한번 press 했을 때	>>> key를 released 했을 때

>>> key를 한번 누르는 순간 timer control register를 stop 상태로 만들고 설정해준 3개의 무한루프 중 첫번째 무한루프를 돌고 있는 것을 확인할 수 있다. 만약 key가 release되면 첫번째 while문을 빠져나와 두번째 while 문으로 가서 다시 key가 눌리는 순간을 기다리게 된다. 즉, key data register를 polled io하게 된다. 이대로 key를 누르지 않으면 계속 무한루프를 돌며 stop 상태를 유지하게 된다.

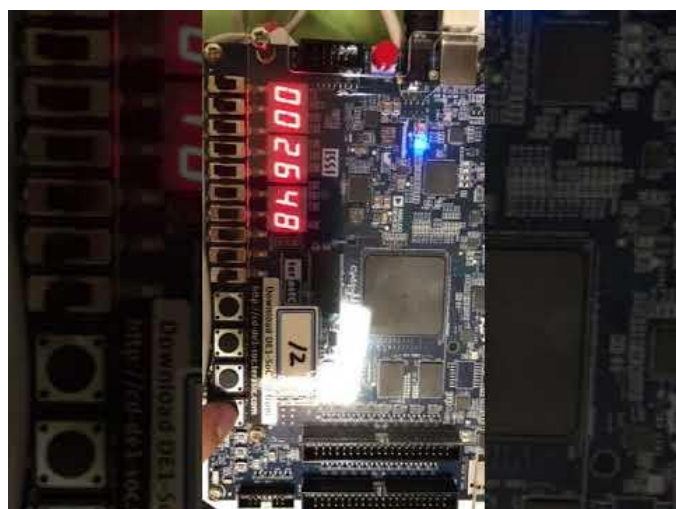
전자 HW 설계 – 실습 보고서

0x000004E4	008FC834	orhi r2, zero, 0xFF20	0x00000368	DEC00104	addi sp, sp, 0x4
0x000004E8	10801404	addi r2, r2, 0x50	0x0000036C	F800283A	ret
0x000004EC	10800037	ldwio r2, 0(r2)			int DIVIDE_millicsec_1(int number) {
0x000004F0	103FFC1E	bne r2, zero, -0x10 (0x000004E4)			DIVIDE_millicsec_1:
		while(1){			addi sp, sp, -0x4
		if(*KEY_ptr!=0) {break;}	0x00000370	DEFFFF04	stw ra, 0(sp)
0x000004F4	008FC834	orhi r2, zero, 0xFF20	0x00000374	DFC00015	int value;
0x000004F8	10801404	addi r2, r2, 0x50			value = (number % 6000 % 100);
0x000004FC	10800037	ldwio r2, 0(r2)			addi r5, zero, 0x1770
0x00000500	103FFC26	beq r2, zero, -0x10 (0x000004F4)			...
		while(1){	0x00000378	0145DC04	...
		if(*KEY_ptr!=0) {break;}			...
0x00000504	008FC834	orhi r2, zero, 0xFF20			...
0x00000508	10801404	addi r2, r2, 0x50			...
0x0000050C	10800037	ldwio r2, 0(r2)			...
0x00000510	103FFC1E	bne r2, zero, -0x10 (0x00000504)			...

>>> 다시 key를 press 했을 때

>>> 다시 key를 release 했을 때

>>> 다시 key를 누르면 3번째 while문으로 가서 key가 release되는 순간을 기다리게 된다. Key가 release된다면 main함수의 while(1)문을 수행하게 된다. 즉, 다시 timer의 status register를 polled io 하게 된다.



해당 프로그램을 보드에서 실행한 결과는 영상으로 게시하였습니다.

<https://www.youtube.com/watch?v=1iE64ITWYVI>