

강의명 : 마이크로프로세서

실습 번호 : 5

실습 제목 : UART programming

학생 이름 : 이학민

학번 : 201910906

1. UART 입출력 함수

1.1

```
int putc(int c)
{
    if ((c=='\n')||(c=='\r'))
    {
        while ((UART0->S1 & TDRE) == 0);
        UART0->D = '\r';
        while ((UART0->S1 & TDRE) == 0);
        UART0->D = '\n';
    }
    else
    {
        while ((UART0->S1 & TDRE) == 0);
        UART0->D = (unsigned char) c;
    }
    return c;
}
```

<프로그램 설명>

TDRE는 0x80u이다. TDRE 변수와의 비트 연산자 계산을 통해 UART0 상태 레지스터의 bit7 값을 읽을 수 있다. bit7은 TDRE로 이 값이 무엇이나에 따라서 전송을 할지 말지를 결정할 수 있다. UART0 상태 레지스터의 bit7은 사전에 만들어져 있는 구조체를 통해 접근하고 접근 방식은 UART0->S1과 같다. UART0->S1과 TDRE를 &계산하여 0이 나오면 transmit queue에 데이터가 가득 차있어서 다른 데이터를 전송할 수 없는 상태이다. 따라서 transmit queue에 데이터가 없어질 때까지 즉, UART0->S1 & TDRE의 값이 1이 될 때까지 while문을 반복시키며 기다린다. 그리고 루프문을 벗어나는 조건을 만족할 때 UART0 데이터 레지스터에 원하는 값을 넣는다. UART0 데이터 레지스터는 사전에 만들어져 있는 구조체를 통해 UART0->D 의 방식으로 접근할 수 있다. 사용자가 입력받은 값인 c가 만약 '\n' 또는 '\r'라면 조건문을 통해 UART0->D에 '\r' 및 '\n'을 넣어주고 '\n' 또는 '\r'가 아니라면 입력받은 c를 부호가 없는 char형으로 변환하여 UART0->D에 넣어준다. 그리고 c를 반환하며 함수를 종료한다.

```

int getc(void)
{
    unsigned int c;

    while ((UART0->S1 & RDRF) == 0);
    c = UART0->D;

    putc(c);
    return (int) c;
}

```

<프로그램 설명>

getc 함수는 한 문자를 입력하고 그 문자를 echo back하는 함수이다. RDRF는 0x20u이다. RDRF 변수와의 & 계산을 통해 UART0 상태 레지스터의 bit5의 값, 즉 RDRF의 값을 읽을 수 있다. 상태 레지스터의 RDRF의 값이 0이면 receive queue에 데이터가 비어있다는 뜻으로 데이터를 받지 않았다는 뜻이다. 따라서 UART0->S1의 bit5의 값이 1이 될 때까지 즉, UART0->S1 & RDRF의 값이 1이 될 때까지 while문을 통해 기다렸다가 조건이 만족되면 부호없는 정수형으로 선언된 c에 UART0 데이터 레지스터의 값을 불러와 대입한다. 앞서 프로그램한 putc 함수를 불러와 c를 출력하고 그 문자를 int형으로 형변환 하여 return 한 후 함수를 종료한다.

```

int getc_nb(void)
{
    unsigned int c;

    if ((UART0->S1 & RDRF) == 0)
        c = EOF;
    else
        c = UART0->D;

    return (int) c;
}

```

<프로그램 설명>

getc_nb는 getc의 함수와 유사하지만 UART0 상태 레지스터의 RDRF 값과 상관없이 기다리지 않고 문자를 입력하는 함수이다. getc는 UART0 상태 레지스터의 RDRF 값이 0이면 1이 될 때까지 기다렸지만 get_nb는 c에 EOF(=-1)을 대입한다. UART0 상태 레지스터의 RDRF 값이 1이면 부호가 없는 정수형 변수로 선언된 c에 UART0 데이터 레지스터의 값을 UART0->D를 통해 불러와 대입한다. 이 함수는 c를 int형으로 형변환 한 후 return하며 종료된다.

```

unsigned int gethex(void)
{
    unsigned char hex[9];

    for(int i=0; i<9; i++)
        hex[i] = getc();

    unsigned int n=0, dec=0;

    for(int i=7; i>=0; i--)
    {
        unsigned int p=1;
        for(int j=0; j<n; j++)
            p *= 16;

        char q = hex[i];
        if(q >= 48 && q <= 57)
            q -= 48;
        else if(q >= 97 && q <= 102)
            q -= 87;
        else if(q >= 65 && q <= 70)
            q -= 55;

        dec += p*q;
        n++;
    }
    return dec;
}

```

<프로그램 설명>

우선 문자 8자리를 받아야 하는데 문자열 끝에는 NULL문자가 포함되어 있으므로 배열의 크기를 9로 선언한다. 총 8자리의 문자를 for문과 getc()를 통해 하나씩 입력받는다. n은 16을 총 몇 번 제공할지 결정하는 변수이고 dec는 10진수 숫자로 바뀐 값을 저장하는 변수이다. 문자 8자리를 NULL문자를 제외한 끝에서부터 읽고 q에 저장한다. 문자 '0'~'9'의 아스키코드는 48이므로 얻은 문자 값에서 48을 빼주어 10진수 숫자(0~9)로 변환한다. 알파벳 'a'~'f'와 'A'~'F' 또한 각각 87, 55를 빼주어 10진수의 숫자(10~15)로 변환한다. 10진수 숫자로 변환된 q에 for문을 반복해 얻은 자릿수에 맞는 16의 제곱수 p를 곱하면 한 자리에 해당하는 10진수의 값을 얻을 수 있다. 제곱수는 자릿수가 하나 왼쪽으로 이동할 때마다 n을 1씩 증가시키며 for문을 여러번 반복하게 함으로써 구할 수 있다. 총 8자리이므로 8번 반복하여 각 자릿수에서 나온 모든 값을 합치면 최종적인 10진수 숫자를 도출할 수 있다. 이렇게 얻어진 10진수 숫자를 return 하며 함수를 종료한다.

```

char *gets(char *str)
{
    char *addr = str;
    int i = 0;

    while(1)
    {
        *(str+i) = (char) getc();

        if(*(str+i)==13)
        {
            i++;
            *(str+i) = '\0';
            break;
        }
        else
            i++;
    }

    return addr;
}

```

<프로그램 설명>

인수 *str은 포인터 변수로 시작 주소를 가지고 있다. 이를 char형 포인터 변수인 addr에 저장하며 함수가 시작된다. 포인터와 배열은 유사한 관계를 가지고 있다. 예를 들어 arr[3]은 *(arr+3)으로 표현할 수 있다. 이를 이용하여 배열을 사용하지 않고 포인터만을 이용해 프로그램을 짰다. while(1)을 통해 무한으로 반복하는 루프문을 만든다. 앞서 프로그램한 getc()를 통해 문자를 하나씩 받아오고 이를 char형으로 형변환하여 *(str+i)에 대입한다. 이때 포인터 변수 str은 char형이기 때문에 i는 while문이 반복될 때마다 1씩 증가시켜야 한다. 엔터키의 아스키코드 값은 13이다. 만약 사용자가 문자열을 다 입력하고 엔터키를 입력하면 함수가 종료되어야 한다. 이에 따라 조건문을 if(*(str+i)==13)으로 쓸 수 있다. 엔터키가 입력되면 i를 1 증가시켜 문자열의 맨 끝에 NULL문자를 추가한 후 while 루프문을 빠져나간다. 함수의 인수에 주어진 시작 주소를 포인터 변수 addr에 저장해두었으므로 return addr를 통해 시작 주소를 return 하며 함수를 종료한다. gets함수는 getc함수를 여러번 반복하는 함수라고 할 수 있다.

```

int puts(const char *s)
{
    int i = 0;
    while(1)
    {
        if(*(s+i)=='\0')
            break;
        else
        {
            putchar(*(s+i));
            i++;
        }
    }

    return 1;
}

```

<프로그램 설명>

인수 *s은 char형 포인터 변수이고 const로 선언되어 있으므로 값이 바뀔 수 없다. 위에서 언급했듯이 포인터와 배열은 유사한 관계를 가지고 있다. 예를 들어 arr[3]은 *(arr+3)으로 표현할 수 있다. 이를 이용하여 배열을 사용하지 않고 포인터만을 이용해 프로그램을 짰다. while(1)을 통해 무한으로 반복하는 루프문을 만든다. 사용자가 입력한 문자열 끝에는 NULL이 있고 이는 문자열의 끝을 알린다. 따라서 NULL이 나오기 전까지 앞서 프로그램한 putchar() 함수를 이용하여 문자를 하나씩 출력한다. 포인터 변수 s는 char형이기 때문에 루프문이 반복될 때마다 i를 1씩 증가시킨다. 문자를 하나씩 출력하다 NULL문자('\0')을 만나면 루프문을 종료하고 1을 return하며 함수를 종료시킨다. puts함수는 putchar함수를 여러번 반복하는 함수라고 할 수 있다.

```

void putclear(void)
{
    puts("^[[");
    putc('2');
    putc('J');
    puts("^[[");
    putc('0');
    putc(';');
    putc('0');
    putc('H');
}

```

<프로그램 설명>

화면을 모두 지우는 ANSI escape code는 ESC[2J 이다. 이를 구현하기 위하여 puts와 putc를 반복적으로 사용하였다. 우선 제어문자 ESC를 입력하기 위해서 vi editor를 통해 c파일을 열고 ctrl+v, ctrl+[를 입력하여 시를 만들었다. 화면을 모두 지우기 위해서는 문자열 2J를 입력해야 한다. 이를 putc 함수를 통해 '2', 'J'를 각각 따로 처리하였다. 화면을 모두 지우고 난 뒤 uart-test2.c 파일에서는 puts("=====")가 실행되는데 이 때 커서가 (0,0)에 있지 않아 엉뚱한 곳에서 =====이 출력되었다. 따라서 커서를 원하는 위치(a,b)로 이동시켜주는 ANSI escape code ESC[a;bH를 사용하였다. 이 때 커서가 (0,0)으로 이동해야 하므로 '0', ';', '0', 'H'를 putc를 통해 각각 따로 입력받았다.

```

void putgoto(int x, int y)
{
    puts("^["");
    putc('0');
    putc(';');
    putc('0');
    putc('H');

    for(int i=0; i<x; i++)
    {
        puts("^["");
        putc('1');
        putc('C');
    }

    for(int i=0; i<y; i++)
    {
        puts("^["");
        putc('1');
        putc('B');
    }
}

```

<프로그램 설명>

앞서 설명했듯이 커서를 원하는 위치(a,b)로 이동시켜주는 ANSI escape code ESC[a;bH를 사용하여 커서를 이동시킬 수 있다. 하지만 원하는 위치의 좌표값이 9를 넘어설 때 아스키코드를 10진수에서 문자형 숫자로 변환하면서 심각한 오류가 발생한다. 예를 들어 10진수 아스키코드를 문자형 숫자로 바꿀 때는 48을 더해야 하는데, 10진수 10은 48이 더해지면 문자형 숫자 10이 아닌 :(콜론) 문자로 바뀌어 버린다. 따라서 ESC[a;bH를 통해서는 원하는 좌표로 마음대로 이동할 수 없었다. 따라서 for문을 통해 x, y좌표를 1칸씩 원하는만큼 이동할 수 있도록 프로그램 하였다. 원하는 x좌표에 도달하기 위해서는 오른쪽으로 이동해야 하고 오른쪽으로 n칸 이동하는 ANSI escape code는 ESC[nC 이다. 한편 y좌표의 경우 아래쪽으로 이동해야 하고 아래쪽으로 n칸 이동하는 ANSI escape code는 ESC[nB이다. 1칸씩만 이동할 것이므로 이 프로그램에서는 n=1을 사용하였다.

커서의 위치가 바뀌고 출력을 마치고 나면 다음에 putgoto가 다시 call 되었을 때 원하는 좌표를 정확하게 찾아가기 위하여 커서가 원점(0,0)으로 돌아와야 한다. 위 프로그램은 커서가 현재 위치에서 상대적으로 움직이는 것이기 때문에 커서의 현재 위치가 반드시 원점에서 시작되어야만 한다. 따라서 함수의 맨 처음 부분에 커서가 (0,0)에 위치하도록 프로그램하였다. 원하는 좌표값으로 이동하는 for문이 작동하기 이전에 '0', ';', '0', 'H'를 putc를 통해 각각 따로 입력받아 커서를 (0,0)으로 이동시키는 ESC[0;0H를 구현하였다.

2. UART 입출력 함수 시험 1

2.1

```
#include "frdm_k64f.h"          // include for FRDM-K64F board

int main(void)
{
    unsigned int x;
    char buf[512];

    soc_init();                  // initialize FRDM-K64F board

    puts("=====\\n");
    puts("If you type a character, it will be printed: ");
    gets(buf);
    putc((int) buf[0]);
    printf("\\n");

    puts("If you type a hexa-decimal number, it will be printed: ");
    x = gethex();
    printf("%08x\\n", x);

    puts("If you type a line of characters, it will be printed: ");
    gets(buf);
    printf("%s\\n", buf);
    puts("=====\\n");

    for (;;)                    // waiting in an infinite loop

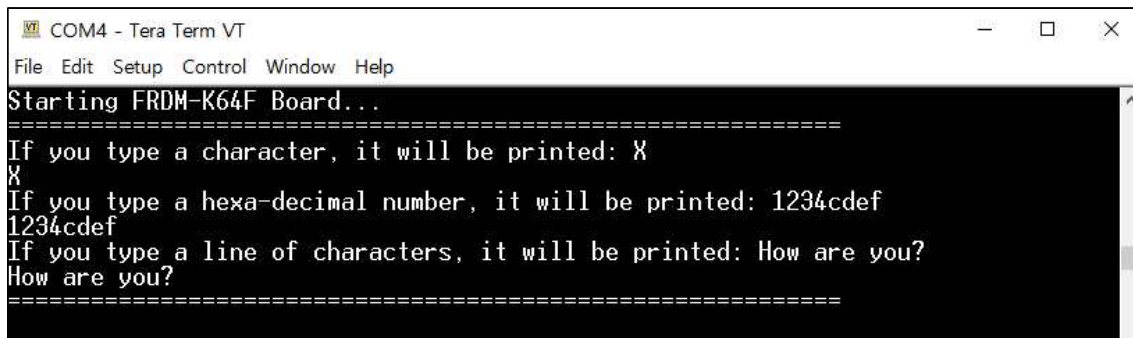
    return 0;
}
```

<c파일 설명>

soc_init();를 통해 보드를 초기화한다. 앞서 프로그램한 puts함수를 불러와 출력하고자 하는 문자열을 출력한다. 앞서 프로그램한 gets함수는 사용자가 문자열을 입력을 할 수 있게 해주는 함수이다. 크기 512의 char형 배열 buf에 문자를 입력하면 buf[0]에 저장된다. putc의 인수는 int형이기 때문에 int형으로 형변환하여 putc((int) buf[0])을 실행하면 사용자가 입력한 문자를 화면에 출력해준다. 16진수형 문자열을 입력하면 앞서 만든 gethex()함수를 불러와 사용자가 입력한 8자리의 문자열을 10진수 숫자로 바꾸어 x에 저장시킨다. 이 때 x는 부호없는 int형 변수로 8자리의 16진수 숫자의 값을 담기에 충분하다. 얻어진 10진수 숫자 x를 printf

와 %08x를 통해 출력시키면 8자리의 16진수로 자동으로 변환되어 출력된다. 따라서 보드에서 실행해보면 16진수 문자열을 입력하면 16진수 숫자가 화면에 출력되는 것을 알 수 있다. 마지막으로 한 줄의 문자열을 입력하면 gets(buf)를 통해 문자열을 엔터키를 치기 전까지 입력 받을 수 있고 이는 배열 buf에 저장된다. printf와 %s를 통해 buf에 저장된 문자열을 한번에 출력한다. for문으로 구성된 무한 루프를 실행시켜 함수가 종료되지 않게 한다. 다만 함수가 종료될 때는 0을 return 하도록 한다.

2.2



```
COM4 - Tera Term VT
File Edit Setup Control Window Help
Starting FRDM-K64F Board...
=====
If you type a character, it will be printed: X
X
If you type a hexa-decimal number, it will be printed: 1234cdef
1234cdef
If you type a line of characters, it will be printed: How are you?
How are you?
=====
```

3. UART 입출력 함수 시험 2

3.1

```
#include "frdm_k64f.h"          // include for FRDM-K64F board

void goto_and_puts(int x, int y, char *str);

int main(void)
{
    soc_init();                  // initialize FRDM-K64F board

    putclear();
    puts("=====\\n");

    goto_and_puts(5, 2, "Are you there!");
    goto_and_puts(10, 5, "I am here!");
    goto_and_puts(15, 8, "Please help me!");
    goto_and_puts(0, 11, "\\n");

    puts("=====\\n");

    for (;;)                    // waiting in an infinite loop

    return 0;
}

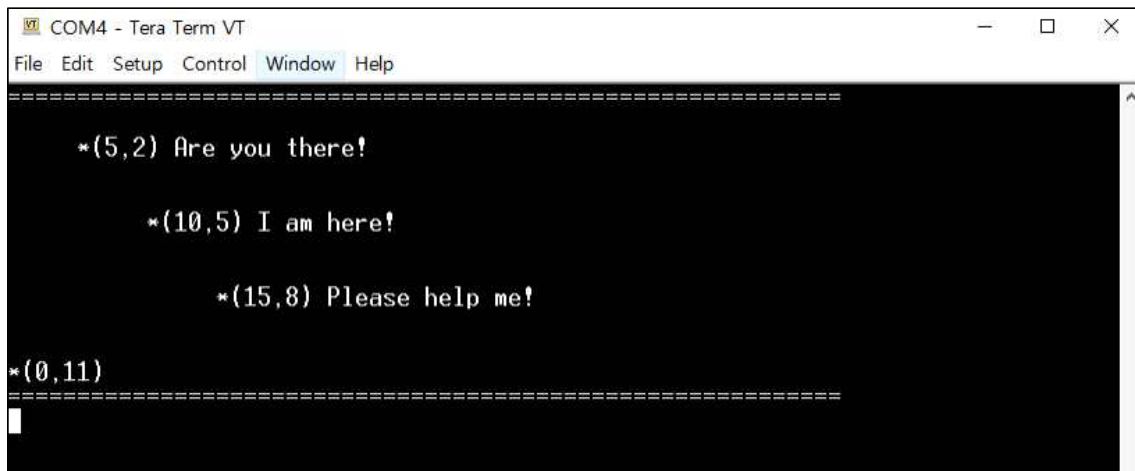
void goto_and_puts(int x, int y, char *str)
{
    putgoto(x, y);
    printf("(%d,%d) %s", x, y, str);
}
```

<c파일 설명>

soc_init();을 통해 보드를 초기화하고 앞서 프로그램한 putclear를 통해 화면 전체를 지운 후 (0,0)의 좌표로 이동한다. (0,0)에 커서가 이동되면 =====를 출력한 뒤 goto_and_puts 함수를 실행시킨다. goto_and_puts는 인수로 이동할 x,y의 좌표값과 출력할 문자열을 갖는다. goto_and_puts 함수는 앞서 프로그래밍한 putgoto(x,y) 함수를 통해 위 c파일에 명시된 좌표로 이동하고 커서를 이동시킨 후에는 출력하고자 하는 문자열을 출력시키는 printf문을 실행하는 역할을 한다. goto_and_puts를 각각의 인수인 x,y의 좌표값과 출력하고자 하는 문

자열에 따라 총 4번 실행한 후 =====를 출력하며 마무리한다. 함수는 종료되지 않아야 하기 때문에 for문으로 만들어진 무한루프를 만들어 return을 방지한다. 다만 함수가 종료될 시에는 0을 return하도록 한다.

3.2



```
COM4 - Tera Term VT
File Edit Setup Control Window Help
=====
* (5,2) Are you there!
* (10,5) I am here!
* (15,8) Please help me!
* (0,11)
=====
```

끝.