

미니 프로젝트

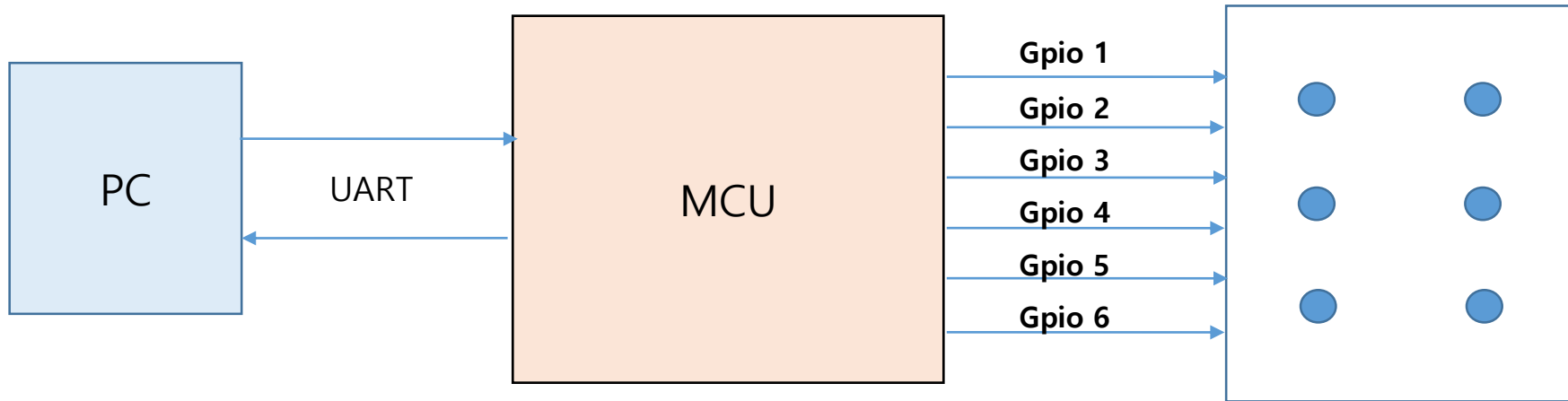
터미널을 이용한 제어

(점자 제어)

김동주, 강민성

• 과 정

- 터미널을 통해 글자 (한글)를 입력한다.
- 입력에 따른 점자를 제어한다. (GPIO 6핀 각각 제어)
- 통신은 UART 통신을 이용한다. (차후에 변경 가능)

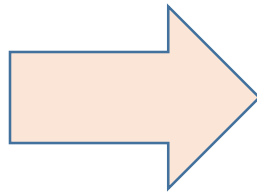


- 주의사항 및 반영할 사항 (기능)

- 오타 시 시각 장애인이 인지할 수 있도록 진동 기능을 추가한다.
(미니 프로젝트시 LED로 구현)
- 입력 받은 글자를 다 읽었을 경우에 다시 데이터를 받을 수 있도록 en신호를 설정한다.
(en신호는 하드웨어 버튼을 눌렀을 시 들어감.)

• 최종 구현 (미니 프로젝트 이후)

- 스마트폰 및 다른 전자기기와 통신 가능하도록 구현한다.
- 영화관 자막을 받아 점자로 표현하도록 구현한다. (미정 사항)
- 웨어러블 및 소형화를 한다. Ex) 장갑, 시계 ...



• 유니코드 및 UTF-8

- 일반 유니코드 - 모든 글자를 2byte로 표현함
- UTF-8 - 유니코드 인코딩 방식 중 하나로써 문자열을 8bit로 저장한다.

ANSI 문자(영어)는 16bit 그대로 저장하며 아시아 문자는 3byte로 가변 표기한다.

Ex) 밑에 사진과 같이 UTF-8 방식으로 출력할 경우..

a는 97 / 가는 234 176 128 / b는 98 / 나 는 235 130 152이 나오는 것을 확인 할 수 있다.

```
setlocale(LC_ALL, "");  
wchar_t a[] = L"a가b나";  
int i;
```

```
wprintf(L"%s\n", a);  
printf("%d\n", a[0]); //a  
printf("%d\n", a[1]);  
printf("%d\n", a[2]);  
printf("%d\n", a[3]); //가  
printf("%d\n", a[4]); //b  
printf("%d\n", a[5]);  
printf("%d\n", a[6]);  
printf("%d\n", a[7]); //나
```

[CortexR5]

97
234
176
128
98
235
130
152
|

- 한글 유니코드의 UTF-8 변환

- 앞 페이지에서 본 가 출력 234 176 128을 기준으로

1110 ^A1010 1011 ^C0000 ⁰1000 ⁰0000

1110 xxxx 10xx xx xx 10xx xxxx (가는 유니코드로 AC00)임

Xxxx 값에 4bit 씩 유니코드 하나씩 대입시켜 주면 UTF-8로 변환 된다.

- 변환 코드 구현

- 가를 string s로 받는다면 $s[0] = 234 / s[1] = 176 / s[2] = 128$ 이 저장됨.
- If 문에서 $s[0]$ 1110 1010 과 $0xE0(1110\ 1010)$ 을 연산해서 UTF-8 한글만 받도록 설정.
 $S[0]$ 1110 1010 과 $0x0F(0000\ 1111)$ 을 연산하여 1010을 살려주어 12bit shift 해줌



15 11 7 3

1010 1100 0000 0000 (가는 유니코드로 AC00)임

이런식으로 s[1], s[2] 부분도 비트연산하여 shift 하게 되어 주면 AC00가 나오게 된다.
따라서 UTF-8 한글 코드를 순수 유니코드로 변환할 수 있다.

1110 1010 1011 0000 1000 0000
1110 xxxx 10xx xxxx 10xx xxxx (가는 유니코드로 AC00)임

```

/ USER CODE BEGIN 1 /
wstring UTF8toUnicode(const string &s)
{
    wstring ws;
    wchar_t wc;

    for(i=0; i<s.length(); i++)
    {
        char c = s[i];
        if((c & 0xE0) == 0xE0)
        {
            wc = (s[i] & 0x0F) << 12;
            wc |= (s[i+1] & 0x3F) << 6;
            wc |= (s[i+2] & 0x3F);
            i += 3;
        }
        ws += wc;
    }
    return ws;
}

```

• 변환 코드 구현 결과

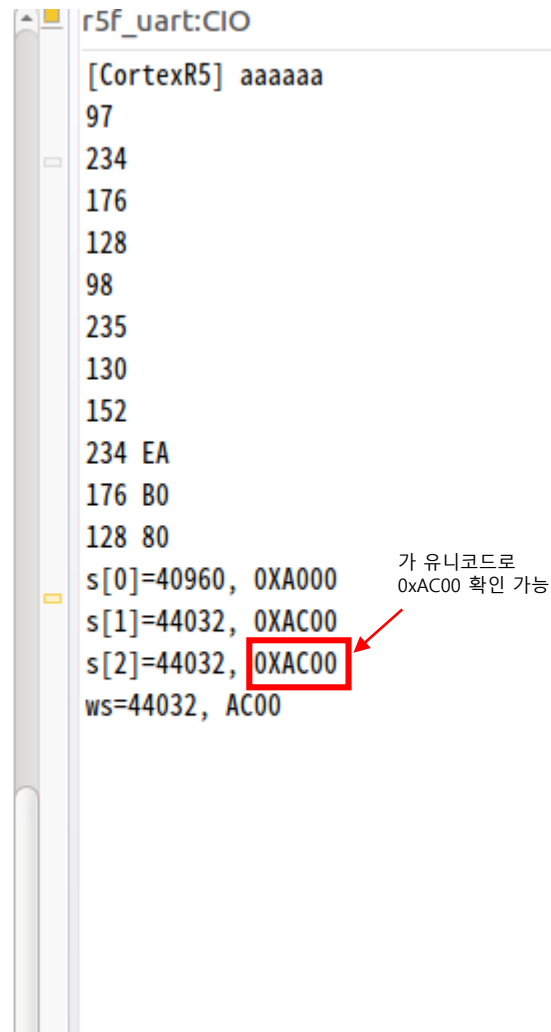
```
wchar_t UTF8toUnicode(const wchar_t *s)
{
    char c = s[0];
    wchar_t *tw;

    printf("%d %X\n", s[0], s[0]);
    printf("%d %X\n", s[1], s[1]);
    printf("%d %X\n", s[2], s[2]);

    if ((c & 0xe0) == 0xe0)
    {
        wc = (s[0] & 0x0f) << 12;
        printf("s[0]=%d, %#X\n", wc, wc);
        wc |= (s[1] & 0x3f) << 6;
        printf("s[1]=%d, %#X\n", wc, wc);
        wc |= (s[2] & 0x3f);
        printf("s[2]=%d, %#X\n", wc, wc);
    }
    ws += wc;

    printf("ws=%d, %X\n", ws, ws);

    *tw = &ws;
    return tw;
}
```



r5f_uart:CIO

[CortexR5] aaaaaa

97

234

176

128

98

235

130

152

234 EA

176 B0

128 80

s[0]=40960, 0XA000

s[1]=44032, 0XAC00

s[2]=44032, 0XAC00

ws=44032, AC00

가 유니코드로
0xAC00 확인 가능

- 진행 상황 (2019. 01. 18)

- 한글 초성, 중성, 종성 함수 작성

- ① 완성형 글자 '각'을 각각의 'ㄱ', 'ㅏ', 'ㅑ'로 분리하여 점자로 구현 할 수 있도록 함수 구현

- 한글, 영어 입력 받아 LED 작동 (솔레노이드 작동에 앞서 LED 작동으로 확인)

- ① 영어 입력 시 LED 작동 확인
 - ② 한글 입력 시 LED 작동 확인 중에 있음

• 문제점

• 현 문제점 (2019. 01. 14)

- ① 리눅스는 UTF8 방식이나 윈도우는 CP949 방식 -> Cp949toUnicode 함수 필요
- ② 영어와 한글을 구별해서 출력 해야함 (유니코드와 아스키코드의 차이) -> 유니코드는 7번째 비트가 1임
- ③ 한글 입력시 LED 작동 시 문제 -> 한글은 영어와 달리 3byte(UTF-8)이라 코드 수정이 필요함

• 예상되는 문제점

- ④ 하드웨어 구현시 솔레노이드 여러 개를 작동시킬 배터리 문제 -> 배터리 선정에 많은 시간 소요
- ⑤ 초, 중, 종성으로 나뉘어 솔레노이드 작동 하려 할 경우 : 순차적으로 코딩처리 시 솔레노이드도 순차적으로 작동할 가능성
-> Interrupt 또는 RTOS 사용 고려

- 현 문제점 해결

- 영어와 한글을 구별해서 출력 (유니코드와 아스키코드 차이)

```
bool chkhan(const char *ch)
{
    if((*ch & 0x80) == 1)
        return true; // 최상위 비트 1 인지 검사
    else return false;
}

bool chkHanUnicode(const uint32 *ch) // for unicode
{
    return !(ch < 44032 || ch > 55199); // 0xAC00(가) ~ 0xD7A3(힐)
}
```