embeded system report

Deadline 2020

[201601639 컴퓨터공학부 홍승현] e-mail: cx\_nf@inu.ac.kr Contact : 010 – 4019 - 2514

목차

[Lab3 1](#_Toc42807332)

[서론 2](#_Toc42807333)

[보고서 설명 2](#_Toc42807334)

[세팅 3](#_Toc42807335)

[내용 3](#_Toc42807336)

[INPUT OUTPUT 설정 3](#_Toc42807337)

[Input PIN 설정 3](#_Toc42807338)

[TFT-LCD PIN 설정 4](#_Toc42807339)

[BREADBOARD 4](#_Toc42807340)

[설계 전 초기 접근 방법 6](#_Toc42807341)

[구조체 세팅 설명 7](#_Toc42807342)

[내용 7](#_Toc42807343)

[TFT-LCD-Info 구조체 7](#_Toc42807344)

[Cursor 구조체 8](#_Toc42807345)

[Button 구조체 8](#_Toc42807346)

[Buffer 구조체 9](#_Toc42807347)

[Screen 구조체 10](#_Toc42807348)

[설계 12](#_Toc42807349)

[설명 12](#_Toc42807350)

[열거형 12](#_Toc42807351)

[구현 14](#_Toc42807352)

[설명 14](#_Toc42807353)

[초기 실행단계 15](#_Toc42807354)

[Cursor 21](#_Toc42807355)

[버튼 입력 27](#_Toc42807356)

[버튼 처리 28](#_Toc42807357)

[출력 처리 34](#_Toc42807358)

[에러핸들링 36](#_Toc42807359)

[설명 36](#_Toc42807360)

[커서 이동 에러 핸들링 36](#_Toc42807361)

[줄 사이의 이동 36](#_Toc42807362)

[커서 이동시 발생하는 커서 잔존현상 40](#_Toc42807363)

[값 INSERT 오류 핸들링 41](#_Toc42807364)

[화면 이동 에러 핸들링 42](#_Toc42807365)

[추가 구현 45](#_Toc42807366)

[설명 45](#_Toc42807367)

[커서의 종류 45](#_Toc42807368)

[포커싱 되어있을 때 커서 버튼에 따른 커서 이동 45](#_Toc42807369)

# Lab3

TFT-LCD를 이용한 편집기 프로그램을 구현

* 현재 입력중인 문자 밑에 커서가 깜빡거려야 한다.
* 일반적인 편집기에서의 Insert모드로 구현해야 한다.
* 일반적인 편집기에서의 Delete 모드를 구현해야 한다.
* 버튼 당 세 개의 문자가 할당되는데 버튼을 누를 때마다 값이 변경되어야 한다.

# 서론

## 보고서 설명

TFT-LCD를 이용한 편집기 프로그램 구현설명을 목적으로 두었다. 세팅, 접근방법, 선언, 구현방법순으로 열거하고 후에 모든 요구조건들을 세세히 기술하여 작성하였으며 구현방식이 어떻게 되어있는지를 보이기 위해 순서도를 중점적으로 사용하여 이해도를 높였다.

# 세팅

## 내용

편집기 프로그램을 사용하기 위한 세팅방법에 대한 설명

### INPUT OUTPUT 설정

편집기 프로그램 설정에 필요한 Button INPUT과 TFT-LCD PIN을 각각 밑 설명과 같이 세팅한다. 그 중 TFT-LCD PIN에 해당하는 Target은 점프선을 통해 TFT-LCD에 연결한다.

### Input PIN 설정

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 첫 번째 버튼  (.qz) | 두 번째 버튼  (abc) | 세 번째 버튼  (def) | 네 번째 버튼  (ghi) | 다섯 번째 버튼  (jkl) | 여섯 번째 버튼  (mno) | 일곱 번째 버튼  (prs) |
| #97 | #108 | #98 | #100 | #101 | #99 | #83 |

|  |  |
| --- | --- |
| 여덟 번째 버튼  (tuv) | 아홉 번째 버튼  (wxy) |
| #103 | #118 |

#### WiringPi for Odroid C1 에서의 GPIO 값

#97 = 24  
#108 = 23  
#98 = 27  
#100 = 22  
#101 = 21  
#99 = 26  
#83 = 7  
#103 = 6  
#118 = 11

### TFT-LCD PIN 설정

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Key 1  (<=) | Key 2  (DEL) | Key 3  (=>) | Reset | LCD  Introduction control | SPI data input of LCD/Touch Panel | SPI data output of Touch Panel | Serial clock of LCD/Touch Panel | LCD chip selection, low active |
| #87 | #104 | #102 | #116 | #115 | MOSI | MISO | SCLK | CE0 |

#### WiringPi for Odroid C1 에서의 GPIO 값

#87 = 1  
#104 = 4  
#102 = 5  
#116 = 2  
#115 = 3  
MOSI = 12  
MISO = 13  
SCLK = 14  
CE0 = 10

### BREADBOARD

빵판타겟 중 가장 윗칸의 3V3, 5.0를 기본적으로 TFT-LCD에 점프선으로 끼워 넣어야 하고, 그라운드(GND)에 연결해야하는 부분이 여러 곳이 있는데 그 중 필자는 파란색 원에 해당하는 그라운드를 사용하였다(Figure 1-1 참조). 또한 노란색 강조부분에서 Header의 넘버링이 되어있는 것을 확인할 수가 있는데 이는 Figure 1-2, Figure1-3 에서의 빨간 글씨의 넘버링을 참고하여 연결하도록 한다.

|  |
| --- |
|  |
| 빵판 연결 부분<Figure 1-1> |

|  |
| --- |
|  |
| TFT-LCD 연결 부분<Figure 1-2> |
|  |
| 빵판 연결 형태<Figure 1-3> |

## 설계 전 초기 접근 방법

1. TFT-LCD내에 pixel을 이용하여 점을 그리는 방식을 수업을 통해 배웠다. 이를 통해 문자를 출력하기 위해서는 각각의 24x24 크기의 픽셀들에 알파벳이 어떻게 그려지는지를 직접 일일히 그려야 한다. 이 때 커서가 깜빡거릴 때의 부분을 인지하고 그려야 한다.
2. 현재 입력중인 문자 밑에 커서가 깜빡거려야 한다. 또한 커서의 위치가 어떻게 지정되어 그려져야 하는지도 판단해야한다.
3. 커서가 오른쪽 이동을 통해 스크린 바깥으로 넘어가려 할 때, 또는 (0,0)에서 왼쪽으로 이동하려 할 때 어떻게 에러 핸들링 해야 하는지 고민할 필요가 있다.
4. 문자를 출력할 때 기존의 버튼을 눌렀는지 새 버튼을 눌렀는지 확인을 어떻게 할 것인가? 이 때 새로운 값을 기존에 덮어씌우게 두면 안되는데 이는 어떻게 구현할 것인가?

그 외 여러 접근 방법을 구상 및 시도 후 최종적으로 결정한 결과로는 각각의 위치 및 역할에 맞게 구조체를 만들어서 활용하는 방안을 채택하여 설계하였다. 다음은 구상했을 때에 Todo-list를 작성한 commit이다.

구성 전 TODO-List: <https://github.com/WhiteHyun/Embedded-Systems/commit/c4214d80204bba8b9ee67f8ce5f901ffa691f32a>

구상 후: <https://github.com/WhiteHyun/Embedded-Systems/commit/cdbeffb1d7bd47842dbc881d83913a33e8bb9354>

## 구조체 세팅 설명

### 내용

위 설명에서 말한 바와 같이, 각각의 구조체 이름에 걸맞게 그 역할에 따른 구조체를 만들어 두었다. 이 5가지의 구조체 정보를 가지고 TFT-LCD에 대한 전반적인 구성을 담당하게 될 것이다.

### TFT-LCD-Info 구조체

|  |
| --- |
|  |
| TFT-LCD 구조체 요소<Figure 1-4> |

#### 설명

LCD정보를 가지고 있는 fb\_var\_screeninfo 구조체의 fbvar과 정보를 읽어오는fbfd, 그리고 LCD의 주소를 가리키는 pfbdata, 이 세 가지를 기존 TFT-LCD설정에 필요한 데이터이므로 “TFT-LCD-Info” 라는 구조체의 멤버로 할당하였다.

### Cursor 구조체

|  |
| --- |
|  |
| Cursor 구조체 요소<Figure 1-5> |

#### 설명

앞으로 자주 나올 커서의 구조체이다. 이동할 때마다 offset\_x, y의 값이 항상 바뀌며, originalTime과 isFlickering이라는 변수는 커서가 깜빡이는 행위에 대한 활동제어변수라고 보면 된다. 여기서 time\_t의 반환형이 있는데 이를 사용하기 위해서는 라이브러리 <time.h>를 include 하여야 한다. pointer는 해당 커서가 가리키고 있는 값의 인덱스를 나타낸다. 이는 버퍼의 인덱스이며, 버퍼에 대해서는 뒤에 설명하도록 할 것이다.

### Button 구조체

|  |
| --- |
|  |
| Button 구조체 요소<Figure 1-6> |

#### 설명

멤버가 하나밖에 없어서 구조체를 사용하지 않고 하려 했으나, 각각의 역할을 눈으로 보여주기에는 구조체를 사용하는 것이 적합하다 생각하여 망설임 없이 만들었다. 해당 멤버는 버튼 12개에 대한 각각의 wiringPi 넘버링 값을 가지고 있다. 그렇기 때문에 라이브러리 wiringPi.h를 include한다. 여기서 BTN\_SIZE는 12이다.

### Buffer 구조체

|  |
| --- |
|  |
| Buffer 구조체 요소<Figure 1-7> |

#### 설명

한 문자가 입력될 때 그에 대한 정보들을 가진다. 눌려진 버튼 값이 Button 구조체 멤버의 몇 번째 인덱스에 해당하는지를 알려주는 buttonNum  
같은 버튼이 눌려질 때 a->b->c->a->b->c처럼 중첩되서 바뀌게 도와주는 overlap 변수,  
입력된 문자가 focus 가 되어있는지를 판별해주는 focused 변수,  
문자가 입력되어 존재하고 있는지(띄어쓰기 포함)를 판별해주는 valueExist 변수,  
기존 출력된 값에서 그 위치에 값이 변경되었는지 아닌지를 판단해주는 isChanged 변수가 있다.  
조금 더 구체적으로 설명하자면, 어느 문자가 입력되었을 경우 문자에 해당하는 Button 멤버의 buttons의 인덱스를 buttonNum 변수가 가진다.  
그리고 한 버튼에는 세 개의 문자로 구성되어 있기 때문에 버튼이 눌려질 때마다 다른 문자가 나와야 하므로, 중첩변수인 overlap이 그 역할을 도맡는다(0->1->2->0 순으로 바뀌는 것을 체크해줌).  
focused는 커서에 해당하는 문자가 같은 버튼이 눌려 값이 변경될 수 있는 상태가 되어있는지를 확인해주는 변수인지를 나타낸다.  
valueExist는 NULL값을 제외한 값이다. 초기에 버퍼의 정보를 전부 NULL로 초기화를 하는데 이는 다음 그림과 같다.

|  |
| --- |
|  |
| Buffer 구조체 설명<Figure 1-8> |

해당 그림의 각 칸들은 각각의 버퍼 구조체를 가진 배열을 의미한다. 실제 LCD를 출력할 때 이러한 정보를 가지고 출력을 하게 되는데 다음 Screen구조체에서 설명된다.

여기서 기존 값이 초기에 설정되는 값이라고 보면 되며, 값이 들어오는 경우 Buffer의 값이 할당된 값이라는 수로 바뀐다는 것을 볼 수 있다(처리 과정에 대해서는 버튼 처리 함수 설명에서 추가로 설명하도록 하겠다).

여기서 a값은 버튼 인덱스 1에 해당하는 것이고, focused는 입력을 방금 받았으니 true, overlap은 기본, valueExist는 값이 들어갔기에 true, isChanged는 값이 변경되었으므로 true로 바뀌어져 있을 것이다.

### Screen 구조체

|  |
| --- |
|  |
| Screen 구조체 요소<Figure 1-9> |

#### 설명

Screen, 즉 TFT-LCD에 정보를 가지고 출력을 하게될 때 사용하는 구조체이다. 여기서 문자(알파벳, ‘.’)에 대한 정보들이 int자료형으로 선언되어 들어가며, 직전 설명한 Buffer 구조체를 버퍼의 최대 크기만큼을 배열로 해서 가지게 된다.

여기서 NUM\_CHAR은 문자의 개수를 뜻하고(27개).  
CHAR\_PIXEL\_SIZE는 문자에 대한 픽셀의 크기를 뜻하며(24x24 pixel을 가지므로 24로 설정함)  
MAX\_BUF\_SIZE는 최대 입력되는 문자의 수를 의미한다(80자).

이 정보를 가지고 앞으로 설명할 대략적인 그림은 다음과 같다.

|  |
| --- |
|  |
| 대략적인 흐름 표현도<Figure 1-10> |

# 설계

## 설명

이 부분부터 전반적인 세팅과 설계의 내용을 약술할 것이다. 또한 어떻게 동작되는지 간략한 그림을 통해 시각적으로 표현하였다.

## 열거형

설계 설명 전 구조체에 들어갔던 열거형 내부 상수에 잠깐 설명하고자 한다. 상수를 만듦으로써 정확하게 어떤 값이 어떠한 이유로 들어갔는지를 확인할 수 있기에 특정 값을 상수로 만들었다. 이 때 #define매크로를 사용하지 않은 이유는 열거형을 통해 상수가 어느 역할을 하는지 그룹화 할 수 있기에 사용하였다.

#### 버튼 wiringPi상수

|  |
| --- |
|  |
| 버튼 상수 열거형 코드<Figure 2-1> |

##### 버튼 열거형 설명

버튼 WiringPi에 해당하는 수를 상수로 만들어 할당하였다. 버튼 구조체의 값을 할당할 때 버튼 열거형을 가지고 할당하게 될 것이다.

#### 크기 상수

|  |
| --- |
|  |
| 크기 상수 열거형 코드 <Figure 2-2> |

##### 크기 열거형 설명

문자가 최대 몇 자까지 쓰일 수 있는지, 또는 실제 커서의 최대 offset값이 얼만큼 들어갈 수 있는지,  
버튼의 개수는 무엇인지, 등 여러 크기나 개수에 대한 값을 상수형으로 지정하여 size라는 열거형 이름으로 구분 지어 놓았다.

#### 그 외

|  |
| --- |
|  |
| 커서상태 상수 열거형 코드(상), 색상 상수 열거형 코드(왼쪽 하단),  중첩상태 열거형 코드(오른쪽 하단)<Figure 2-3> |

##### 그 외 열거형 설명

커서가 왼쪽으로 이동하는지 오른쪽으로 이동하는 지를 나타내는 상수값, 문자를 어느 색깔로 쓸 것인지 나타내주는 색상 상수값, 버퍼에 문자가 존재하지 않거나, 존재할 때 넣어주는 버퍼상태에 대한 상수값을 정의하였다.

## 구현

### 설명

TFT-LCD Lab에서 생각해야 할 것은 크게 커서, 버튼 처리, 화면 출력로 정의할 수 있다. 이에 대한 전반적인 코드 설명을 리뷰한다.

### 초기 실행단계

|  |
| --- |
|  |
| 코드 실행 흐름도<Figure 2-4> |

#### 설명

전반적인 함수의 흐름도를 그렸다. 일종의 멘트도 구성하였는데, 이해하기 편하도록 만들어보았다.

초반에 TFT-LCD나 버퍼 등 구조체에 대한 전반적인 값들을 초기화 해주는 세팅 구성 단계, 그 후 버튼 입력을 받는 함수에 접근하게 된다. 가장 많은 시간이 버튼 입력에 사용되며(이를 언급한 이유는 커서 설명에 나온다), 버튼을 입력받고 나면 어느 버튼을 받았는지에 대한 정보를 리턴한다.

리턴받은 값을 가지고 버튼 처리 함수를 불러 버튼에 따른 처리를 맡는다. 버튼 처리로 변경된 사항에 대한 정보값을 Screen 구조체에 저장한다.

저장된 Screen 구조체의 정보를 가지고 출력 처리 함수가 실행이 된다. 함수에서는 값이 어떻게 변경되었는지를 보고 그에 따라 출력을 수행하는 역할을 하게 된다.

그 중 해당 파트에서는 세팅 구성에 대한 설명으로 시작된다.

#### TFT-LCD설정 구성, 초기화

앞서 알아본 TFT\_LCD 구조체 “변수”를 메인 함수에 선언하여 TFT-LCD에 입출력이 가능하도록 세팅구성을 한다.

fbfd 변수에frame buffer의 node인 /dev/fb를 오픈하여 결과값을 저장한다.

읽어온 fbfd를 가지고 fb\_var\_screeninfo 정보를 가져와 fbvar에 저장한다.

Mmap을 이용하여 pfbdata의 값을 초기화해준다. 아래 코드는 실제 코드 초기화의 일부를 가져온 것이다.

|  |
| --- |
|  |
| TFT-LCD 초기화 과정 중 일부<Figure 2-5> |

코드는 Init\_TFT에 TFT LCD 정보가 들어가있는 구조체의 주소를 가져와 값을 초기화 시키는 형태로 구성되어있는 것을 볼 수 있다. 다음 그림은 초기화 과정 간략도를 나타낸다.

|  |
| --- |
|  |
| TFT-LCD 초기화 과정 간략도<Figure 2-6> |

아래에 설명되는 초기화 과정도 위와 같이 주소를 참조하여 초기화 하는 방식으로 구성되었으므로 이를 알아두길 바란다.

#### 버튼과 커서 설정, 초기화

버튼은 wiringPi의 번호로 Input 받기 때문에 버튼 구조체의 배열 안에 wiringPi 넘버링에 대한 값이 들어간다(열거형 설명에 잠깐 나왔었다). 그래서 해당 번호에 따른 Input으로 pinMode를 설정하고, pullUpDnControl에 TFT-LCD 버튼은 PUD\_UP, 문자 버튼은 PUD\_DOWN으로 설정하여 버튼 동작을 세팅하였다.

커서의 경우 시작할 때는 왼쪽 위, 즉 (0,0)의 위치에 있게끔 설정하였고, originalTime변수는 코드가 실행되고 있을 때의 시간을 time.h 라이브러리를 이용하여 구한 다음 초기화한다. Pointer또한 0으로 세팅하였다(Pointer는 Screen구조체와 “Relate”하다는 것을 상기하였으면 한다).

|  |
| --- |
|  |
| 버튼과 커서에 대한 초기화 코드<Figure 2-7> |

#### 스크린 설정 초기화

Screen 멤버인 alphabet은 알파벳 픽셀값과 문자 ‘.’의 픽셀값이 들어있는 배열이다. 이 픽셀에 관련해서는 보고서 폴더 중 image에 넣어놓았으며, 값을 토대로 일일히 값을 초기화한다. 밑 그림은 필자가 알파벳을 그리면서 픽셀값을 구하고 있는모습이다(Copy한 것이 아님을 조심스레 주장하며 글을 이어가겠다).

|  |
| --- |
|  |
| 알파벳 픽셀값을 구하고 있는 필자의 모습<Figure 2-8> |

멤버 중 80 크기의 Buffer 배열의 값은 false와 \_null(-1)로 초기화 해주는데 이 값들이 NULL상태임을 나타낸다고 보면 된다.

|  |
| --- |
|  |
| screen구조체변수의 buffer 초기화 코드<Figure 2-9> |

#### 화면 초기화(SetScreen)

화면을 설정해주는 함수이다. 이는 문자를 클리어 하거나 그릴 때, 또는 전체 화면을 초기화 할 때 사용된다.

SetScreen은 네 가지의 매개변수를 받는다.

/\*

 \* 화면 초기화 메소드, cursor의 위치를 가지고 draw함

 \* --First parameter(flag)--

 \* true: 전체 화면 클리어

 \* false: cursor가 가리키는 화면 클리어

 \* --second parameter (alphabet)--

 \* null: 위치된 cursor쪽에 문자를 그림

 \* not null: 알파벳 정보를 가지고 알파벳을 그림

 \* Third parameter: LCD의 정보를 담은 객체 포인터

 \* Fourth parameter: 커서의 정보를 담은 객체 포인터

 \*/

세 번째 파라미터의 설명은 넘어가고, 첫 번째 매개변수인 flag는 bool의 값으로, 참일 경우 TFT-LCD에 출력되어있는 모든 값들을 검은색으로 초기화하며, false일 경우에는 네 번째 매개변수로 받은 커서의 정보를 토대로 값을 출력하는데 이는 두 번째 매개변수의 정보에 따라 문자를 출력할 것인지 띄어쓰기나 NULL과 같이 비어있는 검은 칸을 출력할 것인지를 정한다. 아래 그림을 보면 이해하는데 수월하다.

|  |
| --- |
|  |
| SetScreen함수 <Figure 2-10> |

### Cursor

#### 설계 접근

커서는 문자를 출력하는 것과는 별개로 동작하여 깜빡여야 한다. 즉 버퍼의 값에 들어가지 않고 독립적으로 실행해주어야 한다는 의미이다. 또한 커서를 이동시킬 때 커서의 위치에 따라 다르게 이동시켜주어야 한다. 이를 구현하기 위해서는 커서의 위치, LCD화면의 정보가 필요하며, 커서를 깜빡이기에 적합한 곳인 가장 시간을 많이 소요하는 공간이 어디인지를 확인해야 한다.

#### 커서 깜빡임 함수

실행 흐름도 Figure 2-4를 보면 보다시피 버튼 입력 대기 함수 부문에서 코드 실행점유율을 압도적으로 차지할 것이기에 커서 깜빡임 함수를 버튼 입력 함수 쪽에 할당하고, 버튼 처리 함수, 출력 처리 함수 마지막 부분에 커서 깜빡임 함수를 넣어 값이 바뀔 때마다 커서가 깜빡이게끔 구현해준다.

그리고 커서를 깜빡이기 위해서는 현재 커서의 위치를 가지는 커서 정보(Cursor 구조체)와 커서를 키고 끌 수 있도록 해주는 LCD 정보(TFT\_LCD\_Info 구조체)가 필요하다. 여기서 필자는 추가적으로 Screen 구조체의 정보를 가지고 함수를 구성하였는데 이는 추가구현 파트에 자세히 다루도록 할 것이다.

아래는 커서를 깜빡이게 해주는 함수의 동작 과정을 나타낸다.

|  |
| --- |
|  |
| 커서 깜빡임 함수의 동작 과정<Figure 2-11> |

##### 커서 깜빡임 함수 설명

커서의 일정한 깜빡임을 위해 커서 내부 멤버에 originalTime이라는 time\_t 반환형의 변수를 보았을 것이다. 이 변수를 통해 기존에 커서를 출력했던 시간을 파악할 수 있으며, isFlickering이라는 변수를 통해 커서가 꺼져있는지 켜져있는지를 판별할 수 있다.

이 정보를 토대로 함수 초반부에 현재의 시간을 구하는 코드를 작성하고 기존의 시간과의 차가 1보다 큰지를 확인한다. 만약 1보다 크게 되면 1초가 지나게 된 경우이므로 isFlickering 변수의 상태를 확인하여 키거나 끄게 만든다. 물론 이 때 isFlickering의 값도 바꿔 주어야 한다.

다음은 커서가 어떻게 그려지게 되는지를 나타내는 코드이다.

|  |
| --- |
|  |
| mmap을 이용한 커서 draw 코드<Figure 2-12> |

LCD의 픽셀을 그리는 방식은 mmap을 채택하였다. Pfbdata의 정보를 가지고 임시로 선언한 draw\_offset을 가지고 픽셀을 하나씩 그리도록 구현하였다.

#### 커서의 이동

어느 버튼을 입력하느냐에 따라 이동하는 위치가 나뉜다. 이 때 커서에 위치되어있는 문자가 Focus되어 있는 경우(같은 버튼을 눌렀을 때 변경할 수 있는 경우)에는 커서 이동이 아닌 Focus를 풀어주게 된다.

|  |
| --- |
|  |
| 커서 이동함수 실행 과정<Figure 2-13> |

##### 커서 이동 함수 설명

커서이동 함수를 수행할 때 왼쪽으로 이동하는지, 오른쪽으로 이동하는지에 대한 flag값을 파라미터로 받고, 커서와 스크린의 정보를 가지고 이동시키게 된다. 이 중 오른쪽 이동을 할 때 포커싱이 되어있으면 포커싱을 해제하고 오른쪽으로 커서를 이동시킨다. 이는 가리키고 있는 문자에서 왼쪽 이동과는 별개로 해당 문자에서의 오른쪽을 가리키고 있어야 하기 때문이다. 이를 코드에서는 If 와 elseif를 적절히 사용하여 이를 다음과 같이 구현하였다

|  |
| --- |
|  |
| 왼쪽 커서 이동 코드 (상)와 오른쪽 커서 이동 코드(하)<Figure 2-14> |

#### 커서 이동으로 인한 띄어쓰기 기능구현

앞 부분에 설명하지 않은 띄어쓰기에 대해 이 부분에서 설명되어진다(Figure 2-7 참조).

지금까지의 커서 이동은 값에 관계없이 커서의 이동만을 보았다면 지금 볼 것은 현재 커서의 값 존재 유무 + 뒤에 값이 존재하지 않는 상태에서 커서 이동을 하게 되는 경우를 볼 것이다.

LCD에 출력될 모든 값들은 Buffer라는 구조체에 저장되어있고 Buffer는 Screen에서 배열의 멤버로 할당되어있다(Figure 1-7, 1-8, 2-4 참조).  
그리고 기본적인 세팅구성을 할 때, Screen 구조체 내의 버퍼값은 모두 값이 없는 상태(NULL)로 초기화를 해주게 된다.  
여기서 여러 값들이 입력되거나 지워지거나 커서가 이동되거나 할 것이다. 이 때 위치된 커서의 값이 존재하지 않으면서 오른쪽 커서를 이동했을 때의 값이 존재하지 않는 NULL상태라면, 오른쪽으로 이동하기 전 공간은 NULL이 아닌 띄어쓰기로 인식을 하게 된다. 마치 ‘ ‘과 ‘\0’은 다른 것처럼 말이다.

|  |
| --- |
|  |
| 띄어쓰기에 대한 설명<Figure 2-15> |

이는 Buffer의 멤버변수인 valueExist를 통해 해결한다. 띄어쓰기는 값은 존재하지만 문자가 입력되는 공간은 아니므로 valueExist를 제외한 값은 초기에 초기화된 값으로 유지된다.

#### 커서 설명 마무리

지금까지 커서의 전반적인 흐름에 대해 살펴보았다. 앞으로 설명할 부분에 대해서 커서는 위치에 맞게 독립적으로 실행된다는 것을 기억하고 읽어갔으면 하는 바람이다.

### 버튼 입력

#### 설계접근

버튼 처리 설명을 하기에 앞서 버튼 입력이 어떠하게 구성되어 있는지 확인해본다.  
대략적으로 버튼을 입력받을 때 여러 방법이 존재하는데 그 중 flag라는 변수를 활용하여 각 비트별 역할을 주어 입력을 받았다. 다음 사진은 입력에 대한 설명도이다.

|  |
| --- |
| ------------------------------------------------------------------------------------------------------    ------------------------------------------------------------------------------------------------------ |
| 입력처리과정 소스코드1(상)  Input함수 도식화(중)  Flag를 이용한 입력과정(하)<Figure 2-16> |

##### 버튼 입력 함수 설명

flag비트는 int형으로 선언되어 있으며 그 중 하위 12비트를 사용하게 된다.

그림을 보면 각각의 비트 위치가 무엇을 나타내는지 표기해 두었다. 그림 기준 하위 비트 0에 해당하는 부분은 오른쪽 커서 이동(=>)을 입력받을 때 값이 설정되는 부분(실제 코드에서는 하위비트를 ‘.qz’로 본다)이며, 12가지 중 하나의 입력이 들어올 경우 해당되는 비트를 1로 변환시켜 저장하게 된다. 이 경우 입력 값을 받았으므로 반복문을 탈출할 수 있게 되는 구조이다.

이 정보에 대한 값을 받아오면, 버튼 처리 함수 쪽에서 입력받은 값을 토대로 어떠한 처리를 할 것이다.

### 버튼 처리

#### 설명

버튼을 입력받고나서 이를 어떻게 처리해서 LCD에 출력하게 할 것인지를 계산하는 함수이다.

각각의 버튼들은 문자, 삭제, 이동과 관련된 버튼들이므로 세 가지로 나누어 처리하였다.

|  |
| --- |
|  |
| 버튼 처리 함수 간략도<Figure 2-17> |

#### 문자 처리

문자를 입력받게 된 경우에 커서에 값이 존재하는지를 판단한다. 커서 부분에 있게 되는 경우에는 다르게 값을 입력해주어야 하기 때문이다. 만약 값이 들어가있지 않는 상태라면 커서 자리에 값을 넣어주면 된다.

|  |
| --- |
|  |
| 버튼 처리 함수 입력 부분 중 값이 존재하지 않는 경우<Figure 2-18> |

이미 값이 존재한다면 한 번 더 조건처리 해주어야 하는 부분이있다. 존재하는 값의 포커싱과 같은 버튼이 눌려져있느냐에 따라서 들어가게 되는 값의 형태가 달라지기 때문에 이를 주의깊게 확인할 필요가 있다.

|  |
| --- |
|  |
| 버튼 처리 함수 입력 부분 중 값이 존재하는 경우<Figure 2-19> |

Insert에 관해서도 포커싱 유무에 따라 다르게 입력받아진다. 다음 그림을 보자.

|  |
| --- |
|  |
| 버튼 처리 함수 Insert 설명 <Figure 2-20> |
|  | |
| Insert 코드 부분 중 1<Figure 2-21> | |

그림 2-20에서 공통적으로 커서 포인터 위치부터 끝까지 값을 한 칸 당겨 넣는다(이에 대한 부분은 그림2-21의 코드로 구현하였다). 그 중에서 포커싱 유무에 따라 값을 뒤에다 넣을지 앞에다 넣을지를 구분지어 놓는다.  
사실상 커서가 가리키고 있는 곳은 포인터가 가리키고 있는 문자공간이므로 포커싱 되어있는 경우에는 커서를 오른쪽으로 한 칸 옮겨 값을 할당해주어야 한다. 이를 토대로 문자처리를 구현하였다.

##### 문자의 정보는 정확하게 어떻게 되어있는가?

문자 입력에 대한 흐름을 알아보았다. 이 파트에서는 문자가 무엇인지를 어떻게 인식하고 값이 들어가는지를 확인해 볼 것이다.

Screen 구조체 내부의 alphabet은 Figure2-22와 같이 저장되어있으며 이는 Lab에서 설명된 버튼의 위치구성과 동일하다. 또한 Figure 2-16을 참고해보면 flag 비트에서도 동일한 것 역시 확인할 수 있다.

Figure2-22를 볼 때, 각각의 세 단어가 하나의 버튼 구성으로 들어가있음을 보인다. 구현방법에서 각 버튼을 누를 때마다 세 단어가 돌아가면서 입력이 되어야 하는데, 버튼의 wiringPi 정보로는 이를 구분하기 힘들다. 그래서 overlap이라는 변수를 버튼정보와 함께 연관되게 사용하여 이를 해결하였다.

|  |
| --- |
|  |
| Screen구조체의 alphabet 변수에 저장되어있는 배열 정보<Figure 2-22> |

|  |
| --- |
|  |
| buttonNum과 overlap 변수 설명 <Figure 2-23> |

#### 커서 처리

커서 처리 부분은 위의 Cursor 구문에 설명되어 있으므로 생략하겠다.

#### 삭제 처리

Delete의 기능은 간단히 커서를 기점으로 뒤에 있는 문자를 지우는 기능이다. 이에 대한 로직은 Insert와 유사하게 구성되어 있다.

|  |
| --- |
|  |
| 버튼 처리 함수 중 Delete설명<Figure 2-24> |

Insert와는 반대로 값이 할당되는 것을 확인할 수 있다. 그리고 나서 존재하는 문자 중 가장 마지막 부분의 문자를 null로 초기화 시켜 delete 수행을 완료한다. 이로써 Delete의 구현은 간단히 마무리된다.

### 출력 처리

지금까지 버튼을 입력받아 그에 따른 버퍼 처리부분을 보았다. 지금부터 볼 것은 버퍼의 정보를 가지고 어떻게 출력이 이루어지는지를 확인해볼것이다.

버튼 처리로 Insert나 Delete코드를 동작하여 값이 바뀌는 경우와 값 변경이 아닌 단순 커서 이동의 경우가 나타날 수 있다. 이는 isChanged 변수와 연관이 깊다. isChanged 변수는 값이 변경됨에 따라 true로 상태값이 변경된다. 이 변수 덕에 출력 처리 구문에서는 처음부터 출력하는 것이 아닌 커서의 포인터를 기점으로 값이 바뀐 지점까지만 출력해주면 되는 것이므로 시스템 오버헤드를 줄일 수 있다.

|  |
| --- |
|  |
| isChanged 변수 설명<Figure 2-25> |

값의 변경상태를 가지고 출력해주면 된다는 것은 알았다. 이제 이것을 가지고 출력을 하여야 하는데 위치를 가리키고 있는 것은 커서밖에 없다. 그렇다면 커서를 직접적으로 변경해서 출력을 하여야하는가에 대한 대답은 “아니다 “이다. 커서를 옮기면서 값을 출력해주고 변경된 상태값은 다시 false로 바꿔주어야 할텐데, 모두 출력하게 되면 커서는 갈 길을 잃게 된다. 기존의 커서 위치가 어딘지를 모르기 때문이다. 여기서 필자가 사용한 해결법은 임시 출력커서를 하나 만들어줘서 실제 커서의 위치의 값으로 초기화해주는 것이다(임시 출력 커서이므로 커서깜빡임과 관련된 변수초기화는 하지 않는다). 이렇게 되면 임시 커서로 변경된 값들을 출력하고 끝내면 실제는 커서 그대로 위치되어 있으니 그만이다. 아래는 코드에 대한 설명이다.

|  |
| --- |
|  |
| 출력 처리 코드 설명<Figure 2-26> |

먼저, isChanged 변수가 true일 때까지 반복한다(뒤의 조건문은 에러핸들링 파트에서 설명하도록 하겠다).

isChanged를 false로 바꿔주고, 띄어쓰기나 NULL로 화면에 출력하지 않는 부분들은 화면 클리어로 대체된다.

문자에 대한 변경사항들이 있으면 그에 따른 alphabet을 출력하기 위해 해당 버퍼의 버튼정보 값에 3을 곱한 후 중첩여부변수와 더한 값을 알파벳 배열의 인덱스로 넣어 참조하여 화면을 출력해준다(3을 곱하는 이유는 그림 2-23을 참고하길 바란다).  
그 후 임시 커서를 오른쪽으로 옮겨 가면서 반복문을 돌게 된다.

# 에러핸들링

## 설명

지금까지 했던 설계에 대한 설명은 전체적인 코드의 흐름을 설명하였지만 에러 처리에 대해서는 따로 언급을 하지 않았다. 에러사항들이 많아 도중에 넣기에는 설명흐름이 복잡해질 우려가 있었기에 에러 처리에 대해서는 이 파트에서 설명될 것이다.

## 커서 이동 에러 핸들링

커서를 왼쪽과 오른쪽을 이동하는 것에 대한 에러 핸들링을 설명한다.

### 줄 사이의 이동

값을 입력하면서 커서가 이동하게 되는데, 이 때 4가지의 문제가 발생하게 된다.

1. 커서가 맨 처음에 위치되어 있을 때 왼쪽 이동을 하는 경우
2. 커서의 위치가 오른쪽 끝에 위치되어 있는데 오른쪽 이동을 하는 경우
3. 커서의 위치가 왼쪽 끝에 위치되어 있는데 왼쪽 이동을 하는 경우
4. 버퍼의 끝을 커서가 가리키고 있을 때 오른쪽 이동을 하는 경우

이 문제에 대해 살펴본다.

#### 커서가 맨 앞(0,0)에서 왼쪽 이동을 하려 할 때

이는 맨 앞에 있기 때문에 움직일 곳이 존재하지 않는다. 그렇기에 아무것도 하지 않도록 코드를 수정해야 한다.

|  |
| --- |
|  |
| 커서 에러 핸들링 처리 중 1<Figure 3-1> |

#### 커서가 LCD Screen에서 가장 오른쪽에 있는데 오른쪽 이동을 하려 할 때

쉽게 생각할 수 있는 오류이다. 이는 위 아래의 공간이 여유로운 상태이므로, 이 경우 다음 줄로 이동 후 x의 위치는 0으로 초기화 하는 코드를 작성하여야 한다.

|  |
| --- |
|  |
| 커서 에러 핸들링 처리 중 2<Figure 3-2> |

#### 커서가 LCD Screen에서 가장 왼쪽에 있는데 왼쪽 이동을 하려 할 때

위의 경우와 반대되는 경우이다. 다시 y의 위치를 내려 저번 줄로 이동한 후 x의 위치는 맨 오른쪽 끝자락에 둔다.

|  |
| --- |
|  |
| 커서 에러 핸들링 처리 중 3<Figure 3-3> |

#### 커서가 화면을 넘어가려 할 때

이 역시 맨 처음 에러 조건과 유사하다. 이미 버퍼의 값이 FULL인 경우이고, 커서는 끝자락을 가리키고 있으므로 다음 위치로 이동할 수 없기에, 아무것도 하지 않는다.

|  |
| --- |
|  |
| 커서 에러 핸들링 처리 중 4<Figure 3-4> |

### 커서 이동시 발생하는 커서 잔존현상

|  |
| --- |
|  |
| 필자가 초기에 겪은 커서 잔존 현상<Figure 3-5> |

위 그림과 같이 커서는 정확하게 나오지만, 커서를 이동할 때 깜빡이던 커서가 꺼지지 않고 그대로 유지되는 것을 볼 수 있다. 단지 커서 깜빡임 함수는 출력만을 제어하는 함수이기에 이를 해결하기 위해서는 추가적인 제어 코드가 필요하다. 필자는 커서가 넘어갈 때 양 옆의 문자를 다시 출력하는 식으로 구현하였다.

#### 커서 이동 시 커서 기준 양 쪽의 문자를 다시 Redraw하는 방식 접근

##### 커서가 마지막 위치나 맨 앞에 있게되는 경우 Redraw 설정

|  |
| --- |
|  |
| 커서 잔존 현상<Figure 3-6> |

물론 커서의 양쪽 구간중에 존재하지 않는 구간이 있다면 출력하지 않는 형식으로 코드를 작성하였다.

각각 커서를 왼쪽이나 오른쪽으로 이동하여 문자를 다시 Redraw하고 돌아오는 것을 볼 수 있다.

### 값 INSERT 오류 핸들링

#### 버퍼 공간의 여유가 없는 경우

값을 넣다보면 어느순간 값의 정점인 80개까지 들어가게 될 수도 있다는 것을 염려해야 한다. 이 때 문자를 더 넣으려고 한다면, 버퍼 배열의 인덱스를 초과하여 out of index 에러가 발생할 수있다. 이를 방지하기 위해서는

초기에 버퍼 공간의 여유가 있는지를 먼저 체크하는 식으로 이를 방지할 수 있다. 버퍼 공간이 남아있는 지는 버퍼의 맨 마지막 부분의 멤버변수 valueExist가 참인지 거짓인지를 보면 알 수 있다.

|  |
| --- |
|  |
| Insert 코드 중 에러 핸들링으로 인해 추가 구현된 부분 <Figure 3-7> |

## 화면 이동 에러 핸들링

스크린에 총 들어갈 수 있는 문자는 77개이다. 하지만 현재 입력받을 수 있는 문자는 최대 80개이므로 추가로 문자를 입력받고 출력시키게 되면 스크린의 바깥을 벗어나기에 에러가 발생하게 된다.

이를 해결하기 위해서는 마치 스크롤을 내리듯이 화면을 한 칸 위로 이동시키거나 아래로 이동시켜야 하며 해당 코드는 화면 출력 함수인 LCDPrint에서 구현된다.

#### 밑으로 화면이 이동되는 경우

이는 화면에 들어가는 77개의 문자가 입력되어 있으면서, 커서의 위치가 가장 오른쪽이면서 가장 밑쪽일 때 발생되어야 한다. 그 외의 경우에는 화면이 밑으로 이동되거나 하는 경우는 일체 없어야 한다. 이를 구현하기 위해 맨 처음의 위치의 값을 첫 버퍼의 값이 아닌 두 번째 줄의 첫 번째 칸의 값을 넣어주며 시작한다. 현 TFT-LCD에서 1줄에 11개의 문자가 들어갈 수 있으므로, 두 번째 줄 시작지점의 인덱스는 11과 같다.   
또한 커서의 위치도 변경시켜야 한다. Figure 3-8의 그림을 보길 바란다. 각 버퍼의 값을 한 줄을 당겨서 출력을 했으므로, 커서 또한 한 줄 밑의 값으로 들어갸아 한다.

|  |
| --- |
|  |
| 화면 이동 (DOWN) 구현 코드 중 1 (상) 간략도(하) <Figure 3-8> |

#### 밑으로 화면이 이동된 상태에서 위로 화면이동하게 되는 경우

그냥 위로 화면 이동되는 경우는 없어야하며, 화면이 밑으로 내려갔을 때만 화면이 위로 올라갈 수 있는 권한이 주어진다. 이러한 조건을 가지려면 커서가 첫 번째 위치에 있을 때 커서의 포인터 값이 0이 아니면 된다. 0이 아닌 경우는 즉 이미 화면이 밑으로 이동된 상태로 봐도 무방하기 때문이다. 이를 이용하여 조건처리 후 커서의 오프셋 값은 스크린을 밑으로 내리는 것과는 반대로 값을 설정해준다. 이는 밑 그림을 참고하길 바란다.

|  |
| --- |
|  |
| Insert 코드 중 에러 핸들링으로 인해 추가 구현된 부분 <Figure 3-7> |

# 추가 구현

## 설명

과제에 나와 있는 것과 별개로 시스템적으로 보기에 좋은 것들을 나름대로 구현해보았다.

## 커서의 종류

포커싱 되어있을 때와 아닐 때의 커서 모양을 다르게 구성해보았다.

과제 설명에서는 밑에 커서가 깜빡이고 있어야 한다고 하였으나, 제품이라고 생각하고 사용자 입장에서 보기 편하게 할 수 있는 것이 무엇이 있는지를 살펴본 결과, 현재 컴퓨터나 휴대폰에 글자를 적을 때 커서가 옆을 깜빡이고 있기에 이를 구현하는 것이 적합하다고 생각하여 구현해보았다.

|  |
| --- |
|  |
| 포커싱 된 커서 (좌), 포커싱이 풀린 커서(우) <Figure 4-1> |

## 포커싱 되어있을 때 커서 버튼에 따른 커서 이동

Figure 4-1에 포커싱 된 커서에서 왼쪽 커서 이동 버튼을 누르면 Figure 4-1의 오른쪽 그림처럼 나타내어지고, 오른쪽 커서 이동 버튼을 누르면 커서가 오른쪽을 향해 있게 구현해두었다.