**CLCD Lab Project (중간고사)**

학과 : 전자공학과

학번 : 201501149

성명 : 이성범

**목차**

**1. Project(과제) 설명**

1.1 서론

1.2 본론

1.3 결론

**2. Project Flow**

**3. Algorithm**

3.1 start

3.2 Input

3.3 Parsing

3.4 Calc

**4. Exception Handling**

4.1 Over flowing Handle

4.2 Invalid Operation Handle

4.3 First Minus Handle

**5. Hardware 설계 (참고)**

**1. Project(과제) 설명**

Odroid-C1 board를 기반으로 Swith를 이용해서 CLCD를 제어하는 프로그램

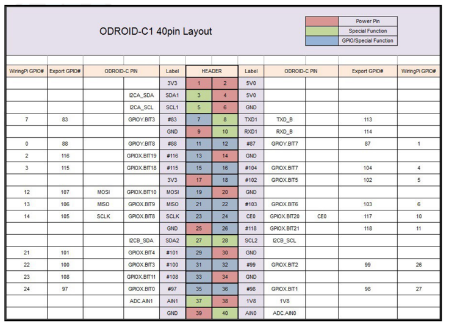
- 스위치로 숫자 버튼, +, -, = 연산자를 입력키로 구현한다.

- CLCD에는 계산기와 같이 디스플레이 되도록 구현한다. ex) 1+2=3

- 개발환경은 리눅스로 minicom을 통해서 개발한다.

- WiringPi의 LCD library를 사용하지 않는다.

1.1 서론

- Odroid Pin map과 wiringPi 에서 제공해주는 Pin map을 보고 매칭시켜서 Hardware 설계를 한다.

- Hardware 설계 후 기본적인 헤더파일 선언과 핀매핑을 define을 통해서 다루기 쉽게 변수를 선언한다.

- DD RAM , CG RAM 두가지 내부 메모리를 사용하기 위해서 기본 세팅을 하는 함수를 사용한다.

-

1.2 본론.

- CLCD를 사용하기 위해서 화면을 초기화 시키고 PinMode를 전부 기본적으로 세팅한다.

- Switch로 Input을 받는 함수를 선언 및 구현한다.

- Switch로 Input 받는 값들을 화면에 출력하고 char형으로 buf에 저장한다.

- EQL ( = ) button이 눌렸을 때 buf에 있는 값들은 number, operator로 나누어 각 각 Parsing이 되어 들어가는 Parsing함수를 선언 및 구현한다.

1.3 결론

- Parsing이 된 후에 number, operator 배열에 있는 값들로 Calculate하고 결과를 화면에 디스플레이 한다.

- 숫자 버튼을 다시 누르면 재시작을 할 수 있게 프로그램을 구현한다.

- 잘못된 수식이나 Overflow를 방지하기 위한 예외 처리 코드를 작성한다.

**2. Project Flow**

|  |
| --- |
| 1. 헤더파일 선언, LCD, Switch Pin mapping |
| 2. Data register, Instruction Register, DD RAM , CG RAM control 함수 선언 |
| 3. CLCD 초기화, 재시작 시 변수, 메모리 초기화(init 함수) |
| 4. Input, Parsing, Calculator 함수 |
| 5. main 함수, start 함수 |

1. 헤더파일 선언, (LCD, Switch) Pin mapping

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <wiringPi.h>

-> 위와 같이 기본적인 헤더파일로만 사용했습니다.

#define LCD\_D4 2 #define LCD\_D5 3 #define LCD\_D6 1 #define LCD\_D7 4

#define LCD\_RS 7 #define LCD\_EN 0 #define BT0 12 #define BT1 13

#define BT2 14 #define BT3 21 #define BT4 22 #define BT5 23

#define BT6 24 #define BT7 10 #define BT8 11 #define BT9 26 #define EQL 27

#define PLUS 5 #define MINUS 6

-> Pin mapping 세팅

1. Data register, Instruction Register, DD RAM , CG RAM control 함수 선언

void write4bits(unsigned char command)

{

digitalWrite(LCD\_D4, (command & 1));

command >>= 1;

digitalWrite(LCD\_D5, (command & 1));

command >>= 1;

digitalWrite(LCD\_D6, (command & 1));

command >>= 1;

digitalWrite(LCD\_D7, (command & 1));

digitalWrite(LCD\_EN, 1);

delayMicroseconds(10);

digitalWrite(LCD\_EN, 0);

delayMicroseconds(10);

}

-> 사용하는 D7, D6, D5, D4로만 write할 수 있게 register에 Write해주는함수

void sendDataCmd4(unsigned char data)

{

write4bits(((data >> 4) & 0x0f));

write4bits((data & 0x0f));

delayMicroseconds(100);

}

-> register에 write해주기 위해서 상위 4비트, 하위 4비트를 나눠서 보내주는 함수

void putCmd4(unsigned char cmd)

{

digitalWrite(LCD\_RS, 0);

sendDataCmd4(cmd);

}

// 상태를 RS : 0로 바꾸면서 Instruction control을 하는 함수

void putChar(char c) {

digitalWrite(LCD\_RS, 1);

sendDataCmd4(c);

}

// 상태를 RS : 1로 바꾸면서 control이 아닌 data를 직접 쓰는 함수

1. CLCD 초기화, 재시작 시 변수, 메모리 초기화(init 함수)

putCmd4(0x28); // 4비트 2줄

putCmd4(0x28); putCmd4(0x28);

putCmd4(0x0e); // 디스플레이온 커서 온

putCmd4(0x02); // 커서 홈

delay(2);

putCmd4(0x01); // 표시 클리어

delay(2);

-> putCmd4(0x28)은 확실히 작동하게 하기 위해서 3번 실행시킵니다.

1. Input, Parsing, Calculator 함수

int Input()

{

int i;

while (digitalRead(EQL) == 0)

{

if (cursor > 32) return 0; // 입력 overflow 발생

if (digitalRead(BT0) == 1)

{

buf[cursor++] = '0';

putChar('0');

delay(500);

}

-> Input 함수 code의 일부로 EQL 버튼이 눌릴 때 까지 switch로 Input을 받습니다.

int Parsing()

{

for (i = 0; i < len; i++)

{

if (i == 0 || (buf[i] >= '0' && buf[i] <= '9')) // 숫자이면

{

num\_buf[num\_len++] = buf[i];

}

else { // 연산자인 경우

// 그 전까지의 number를 버퍼에 넣어준다.

if (buf[0] == '+' || buf[0] == '-')

{

return 0;

}

if (num\_len >= 2)

{

number[num\_idx++] = atoi(num\_buf);

}

else if (num\_len == 1) {

number[num\_idx++] = num\_buf[0] - '0';

}

num\_len = 0;

// 연산자 처리

operate[op\_idx++] = buf[i];

}

-> Parsing 함수 code의 일부로 number, operate 배열에 숫자, 연산자를 넣습니다.

int Calc(int number[], char operate[])

{

int i;

int result = 0;

result = number[0];

for (i = 0; i < op\_idx; i++)

{

if (operate[i] == '+')

{

result += number[i + 1];

}

if (operate[i] == '-')

{

result -= number[i + 1];

}

}

return result;

}

-> Clac 함수로 number, operate 배열에 있는 값으로 계산을 해서 값을 반환합니다.

1. main 함수, start 함수

int main(int argc, char\*\* argv) {

wiringPiSetup(); // wiringPi로 핀을 사용하겠다는 시작의미

initialize\_textlcd(); // LCD 세팅

start();

}

-> main함수는 간단하게 setup, clcd화면 초기화, start함수로 구성되었습니다.

-> start함수는 Input, parsing, init 함수를 실행하는 함수로 알고리즘 파트에서 자세한

설명을 하겠습니다.

**3. Algorithm**

3.1 start

void start()

{

int i;

int result;

while (1)

{

int check\_over = 0; // overflow이면 화면 새로운 창이된 후 바로 숫자를 쓸 수 있게 해줌

char s1[MAX\_SIZE];

result = 0;

if (Input() == 0)

{

print\_overflow();

init(1);

}

else

{

// = (EQL) 버튼을 누른 경우

result = Parsing();

if (result == 0 && error == 1) // number, operate 배열에 각 각 숫자와 연산자가 들어간다.

{

// 올바르지 않은 수식인 경우

print\_invalid();

init(1);

}

else {

sprintf(s1, "%d", result);

int s1\_len = strlen(s1);

for (i = 0; i < s1\_len; i++)

{

if (cursor + i == 16) putCmd4(0xC0);

if (cursor + i > 32) { // 출력 중 overlfow가 발생하는 경우

print\_overflow();

check\_over = 1;

break;

}

putChar(s1[i]);

}

if (check\_over) init(1);

init(2);

}

}

}

}

- Start 함수 설명 -

1. 반복적인 실행을 하기위해서 while(1) 내부에 Input, Parsing, 받은 결과를 출력 시키는 마지막 else 파트를 볼 수 있습니다.

2. Input함수에서 0값이 반환 되지 않았을 때(Overflow 발생하지 않았을 때) 다음 Parsing 함수를 실행합니다.

3. Parsing 함수에서 0값이 반환 되지 않았을 때(Overflow 발생하지 않았을 때) 받은 반환값(결과값)을 result 변수에 할당합니다.

4. sprintf(s1, "%d", result); 함수를 이용해서 s1 버퍼에 result 정수형을 char\* 형으로 바꿔서 할당합니다. 그 후에 결과값을 출력합니다.

3.2 Input

int Input()

{

int i;

while (digitalRead(EQL) == 0)

{

if (cursor > 32) return 0; // 입력 overflow 발생

if (digitalRead(BT0) == 1)

{

buf[cursor++] = '0';

putChar('0');

delay(500);

}

if (digitalRead(BT1) == 1)

{

buf[cursor++] = '1';

putChar('1'); // 1 char는 1 byte 즉, 8bit로 되어있으니까

delay(500);

}

.

. (2 ~ 8 중복으로 생략)

.

if (digitalRead(BT9) == 1)

{

buf[cursor++] = '9';

putChar('9');

delay(500);

}

if (digitalRead(PLUS) == 0)

{

buf[cursor++] = '+';

putChar('+'); // 1 char는 1 byte 즉, 8bit로 되어있으니까

delay(500);

}

if (digitalRead(MINUS) == 0)

{

buf[cursor++] = '-';

putChar('-'); // 1 char는 1 byte 즉, 8bit로 되어있으니까

delay(500);

}

if (cursor == 16) putCmd4(0xC0);

}

putChar('='); cursor++;

return 1;

- Input 함수 설명 –

1. EQL(=) 스위치가 눌리지 않을 때까지 buf[] 배열에 값을 담고 cursor++를 통해서 cursor 위치를 1씩 이동시킵니다.

2. swich를 눌렀을 때 여러 번 인식하지 않게 delay(500)을 통해 시간에 여유를 줍니다.

3. if (cursor == 16) putCmd4(0xC0); 행을 통해서 cursor가 16칸에 값을 출력시켰을 경우에 행을 바꿉니다.

4. if (cursor > 32) return 0; 행을 통해서 32번째 커서까지 입력을 받습니다.

3.3 Parsing

int Parsing()

{

int number[MAX\_SIZE / 2];

char operate[MAX\_SIZE / 2];

int len = strlen(buf); int i;

char num\_buf[MAX\_SIZE]; // 연산자가 나오기 전까지의 number를 담아주는 char buf

int num\_len = 0;

// 숫자를 받아들인 길이가 2 이상이면 atoi 값 넣기

// 숫자를 받아들인 길이가 1이면 (x - '0') 값 넣기 으로

for (i = 0; i < len; i++)

{

if (i == 0 || (buf[i] >= '0' && buf[i] <= '9')) // 숫자이면

{

num\_buf[num\_len++] = buf[i];

}

else{ // 연산자인 경우

// 그 전까지의 number를 버퍼에 넣어준다.

if (buf[0] == '+' || buf[0] == '-')

{

return 0;

}

if (num\_len >= 2)

{

number[num\_idx++] = atoi(num\_buf);

}

else if(num\_len == 1){

number[num\_idx++] = num\_buf[0] - '0';

}

num\_len = 0;

// 연산자 처리

operate[op\_idx++] = buf[i];

}

}

if (num\_len >= 2)

{

number[num\_idx++] = atoi(num\_buf);

}

else {

number[num\_idx++] = num\_buf[0] - '0';

}

printf("num\_idx : %d\n", num\_idx);

printf("op\_idx : %d\n", op\_idx);

if (op\_idx >= num\_idx) return 0;

else

{

return Calc(number, operate);

}

}

- Parsing 함수 설명 –

1. buf에 있는 값들을 for문을 통해 전부 접근합니다.

2. 숫자인 경우 임시로 num\_buf에 char형으로 담아 둡니다. num\_len으로 num\_buf의 현재 길이를 담아 둡니다.

3. 연산자가 나왔을 때 길이가 2이상인경우 num\_buf에 있는 숫자를 atoi 함수로 int로 변환하고 1인경우에 char값 – ‘0’ 을 통해서 number 배열에 담습니다. 그리고 num\_len 값을 0으로 넣고 연산자를 연산자함수에 담습니다.

4. 숫자와 연산자를 담은 배열을 Calc 함수에 넘겨줍니다.

3.4 Calc

int Calc(int number[], char operate[])

{

int i;

int result = 0;

result = number[0];

for (i = 0; i < op\_idx; i++)

{

if (operate[i] == '+')

{

result += number[i + 1];

}

if (operate[i] == '-')

{

result -= number[i + 1];

}

}

return result;

}

- Calc 함수 설명 –

1. result에 number 배열에 있는 값 1개를 담습니다.

-> 첫 번째 입력이 연산자가 아닌 경우에 올바른 수식이라면 number 숫자가 operator 숫자보다 1개 더 많기 때문에 다음과 같이 수행. 첫 번째 입력이 연산자인 경우는 Exception Handling에서 다루겠습니다.

2. result 변수를 피연산자로 operate 인덱스 i번째 +, - 두가지 경우에 따라 number 인덱스 i + 1번째 값으로 연산을 수행합니다.

3. 모든 연산이 끝난 후 result 값을 반환합니다.

4. 그 후에 Parsing 함수에서도 이 값을 반환하게 됩니다.

5. if (result == 0 && error == 1) 이 코드는 result 값이 0인 경우에 invalid\_operation 상태이지만 만약, Clac에서 계산한 결과가 0인 경우에도 invalid\_operation이라고 인식하기 때문에 전역변수 error = 1을 선언하여 clac 함수까지 수행이 된다면 올바른 수식이므로 Clac함수에서 error에 0값을 넣어주면서 해결했습니다.

**4. Exception Handling**

4.1 Over flowing Handle

void print\_overflow()

{

int i;

initialize\_textlcd();

for (i = 0; i < 8; i++)

{

putChar(overflow[i]);

}

delay(2000);

initialize\_textlcd();

}

- 설명 –

1. Overflow가 발생한 경우 화면을 clear하고 char overflow[9] = "Overflow"; 띄운 후 delay(2000)으로 2초 후 다시 화면을 clear합니다.

2. 입력 overflow는 Input 함수에서 if (cursor > 32) return 0; cursor가 32를 넘어가면 발생하게 처리해서 0을 반환하여서 start함수에서 Input이 0인경우 print\_overflow함수를 수행한 후 while문에서 재시작할 수 있도록 continue를 사용하였습니다.

3. 출력 overflow는 start함수에서 if (cursor + i > 32) { print\_overflow();

check\_over = 1;

break;

}

이 부분으로 cursor를 전역변수로 사용해서 현재 CLCD에 있는 커서의 위치와 i가 결과값을 출력하려는 길이(s1\_len)만큼 증가하기 때문에 그 안에 두 합이 32를 넘으면 overflow가 나4게하여 해결 했습니다.

4.2 Invalid Operation Handle

void print\_invalid()

{

int i;

initialize\_textlcd();

for (i = 0; i < 17; i++)

{

putChar(invalid[i]);

if(i == 15) putCmd4(0xC0);

}

delay(2000);

initialize\_textlcd();

}

- 설명 –

1. 잘못된 수식이 발생한 경우 화면을 clear하고 char invalid[18] = "Invalid operation";를 띄운 후 delay(2000)으로 2초 후 다시 화면을 clear합니다.

2. 이 함수 발생 경우는 Parsing 함수에서 number 배열의 길이 -> num\_idx, operate의 길이 -> (op\_idx) 에 담아둔 후에 number의 개수가 연산자보다 적거나 같은 경우에 발생시키도록 if (op\_idx >= num\_idx) return 0; 코드를 통해서 해결했습니다.

4.3 First Minus Handle

- 설명 –

1. **4.2** 예외 처리에서 다른 예외가 생기는데 그 부분은 첫 clcd 입력이 -, + 연산자인 경우입니다. 그런 경우는 연산자와 숫자의 개수가 같아질 수 있기 때문에 Parsing 함수에서 아래 코드와 같이

if (i == 0 || (buf[i] >= '0' && buf[i] <= '9')) // 숫자이면

{

num\_buf[num\_len++] = buf[i];

}

로 코드를 짜서 i가 0인 경우에 그 연산자는 그 숫자의 부호로 사용되도록 하였습니다.

2. 이 상황에서 CLCD 첫번째 두번째 입력에 연산자가 들어가면 잘못된 수식이지만 계산이 그대로 수행되므로

if (i == 1)

{

if (buf[0] == '+' || buf[0] == '-')

{

return 0;

}

}

위 코드를 사용해서 첫번째가 연산자였는데 다음 두번째도 연산자라면 잘못된 수식이므로 0을 return 했습니다.

**5. Hardware 설계 (참고)**

(1) CLCD, CLCD button 설계

주황색 wire : VCC, GND

파랑색 wire : Data

노랑색 wire : CLCD button

|  |  |  |
| --- | --- | --- |
| Pin name | C1/C1/C0+ | wiringPi num |
| LCD\_RS | #83 | #7 |
| LCD\_EN | #88 | #0 |
| LCD\_D4 | #116 | #2 |
| LCD\_D5 | #115 | #3 |
| LCD\_D6 | #87 | #1 |
| LCD\_D7 | #104 | #4 |  |

(2) Switch 11개 설계

BT0 : MOSI -> 12

BT1 : MISO -> 13

BT2 : SCLK -> 14

BT3 : #101 -> 21

BT4 : #100 -> 22

BT5 : #108 -> 23

BT6 : #97 -> 24

BT7 : CE0 -> 10

BT8 : #118 -> 11

BT9 : #99 -> 26

EQL : #98 -> 27

PULS : #102 -> 5

MINUS : #103 -> 6