**Microprocessor Team Project**

**최종보고서**

|  |  |
| --- | --- |
| **조번** | 3조 |
| **주제** | 디지털 사물함식 냉장고 관리시스템 |
| **조원** | 2011253007 김유준 |
| 2012253093 임정재 |
| 2014253048 마영진 |

목차

1. 주제 및 목표

2. 주요 함수 및 핵심 알고리즘

2.1 Main part

2.2 Timer0 PART

2.3 Utility PART

2.4 Data part

2.5 LCD part

2.6 KEYPAD part

2.7 MOTOR part

2.8 UART part

2.9 Python arm kit server part

2.10 Flask web Server part

3. Flow chart

3.1 ARM kit main Flow chart

3.2 Timer0 flow chart

3.3 KEYPAD main Task Flow chart

3.4 Motor main Task flow chart

3.5 LCD main Task flow chart

3.5 UART main Task Flow chart

4. 구현기능 설명

4.1 ARM kit의 구현기능

4.2 flask web server & python arm kit server

5. 조별 일지

6. 고장내역

7. 토의

7.1 구현하지 못한 기능

7.2 계획에 비해 새롭게 추가한 기능

7.3 Final discussion

# 주제 및 목표

## 주제 및 선정 이유

현대 사회에서 냉장고는 필수 불가결한 요소로 주로 음식물 같은 냉장 및 냉동보관이 필요한 물품들을 보관하기 위하여 발명되었다. 일반적으로 일반 가정이나 음식점에서 사용되는 냉장고는 사용하는 사람이 정해져 있기 때문에 별무리 없이 사용 가능하다. 그러나 회사 및 기숙사 등에서 사용되는 공동 냉장고의 경우 불특정 다수가 사용하기 때문에 보관한 음식의 도난을 당할 우려가 존재한다. 또한 다른 음식물과 같이 보관하여 음식물의 변질될 위험 또한 존재한다.

이것을 해결하기 위해 각각의 사용자에게 냉장고를 하나씩 배정하는 방법을 생각할 수 있다. 하지만 이 방식은 보관 효율이 감소하고 관리 및 재정적인 문제를 야기시킨다. 따라서 이 문제를 해결하기 위해 공용 사물함 대여 시스템에서 아이디어를 착안하여 디지털 사물함식 냉장고 관리시스템을 주제로 선정하였다.

디지털 사물함식 냉장고 관리시스템(이하 락커 시스템)은 냉장고에서 일정한 크기의 칸(이하 락커)을 사물함과 유사한 형태로 여러 개를 가지고 있으며 사용자는 대여가능한 락커를 원하는 수만큼 대여하여 사용 및 반납할 수 있도록 하며 락커 이용 시 OTP(one time password)방식의 비밀번호를 제공하여 도난을 방지한다.

## 목표

락커 시스템을 구현하기 위하여 ARM cortex m3 kit(이하 ARM kit)와 Python 및 Flask(Python Microframework)를 이용하여 락커의 대여, 사용 및 반납을 구현하는 것을 목표로 한다.

락커 시스템을 구현하는데 있어서 제한 사항이 존재하며 이를 감안한 구체적인 목표는 다음과 같다.

* 실제 냉장고를 사용할 수 있는 환경이 갖춰지지 않기 때문에 냉장고의 개폐의 구현을 ARM kit의 모터를 회전하는 것으로 확인한다.
* 외부 인터넷에서 접근이 가능한 서버를 소유하지 않아 서버의 대용으로 내부 네트워크를 사용한다.
* ARM kit로 통신을 하는데 있어서 이더넷을 사용하기위한 라이선스 등록에 복잡성이 프로젝트 진행에 차질을 주므로 UART를 이용한 시리얼 통신으로 구현한다

# 주요 함수 및 핵심 알고리즘

## Main part

프로그램의 메인 코드이다. 먼저 프로그램에서 사용되는 Timer0, LCD, KEYAPD, MOTOR, UART를 모두 초기화한다. 각 모듈들은 서로 하나의 GPIO를 통하여 작동하기 때문에 한 순간에 하나의 모듈만이 작동 가능하다. 따라서 무한 루프를 이용함으로써 각 모듈의 mainTask함수들을 실행하여 동시에 모듈을 사용하는 것처럼 실행한다.

//사용할 기능들을 모두 초기화한다

void init() {

init\_timer0();

LCD\_init();

KEYPAD\_init();

MOTOR\_init();

UART\_init();

}

//main

int main(void) {

//사용할 모든 기능을 초기화

init();

//무한 루프를 돌면서 각 기능이 동작하게만든다

while (1) {

Keypad\_mainTask();

LCD\_mainTask();

MOTOR\_mainTask();

UART\_mainTask();

}

}

## Timer0 part

Timer0는 이 프로그램에서 사용 되는 타이머는 1초마다 인터럽트가 발생하여 매초마다 필요한 동작들을 수행하도록 하였다

### Timer0 핸들러

매 초마다 내부의 시간을 업데이트, LCD화면 갱신 및 UART 통신에 필요한 카운터를 업데이트한다.

//매 1초마다 인터럽트 핸들러에 진한다

void TIMER0\_IRQHandler(void) {

//timer0의 인터럽트 비트를 클리어하여 다음 1초 후에 진입하도록 만든다

TIM\_ClearIntPending(LPC\_TIM0, TIM\_MR0\_INT);

//현재 시간을 1초가 지난 시간으로 업데이트 한다

DATA\_timeUpdate();

//1초마다 lcd 화면이 갱신이 하도록 설정한다

LCD\_setRefresh();

//arm 프로세서가 서버와 연결되어있다면 연결확인을 위한 카운터를 갱신한다

if (DATA\_isServerConnected())

DATA\_connectionCountDown();

//비밀번호를 확인하기위해 전달한 메세지를 위한 카운터를 갱신한다

if (DATA\_isRequestedPermission())

DATA\_sendingMsgCountDown();

}

## Utility part

Utility의 함수들은 주로 모듈들을 사용하는 GPIO설정을 조작하는 파트이다.

### GIPO설정 저장 및 로드 함수

Arm 프로세서에서 모듈들은 사용하기 위해서는 GPIO를 사용해야한다. 따라서 매 번 사용하는 모듈 종류가 달라지므로 각 모듈에서 사용되는 GPIO 설정을 저장하거나 로드하기 위해서 다음과 같은 함수들을 사용한다.

// arm processor는 하나의 GPIO를 가지고 있다

// arm processor에서 GPIO는 LPC\_GPIO라는 구조체로 표현되고 LPC\_GPIO0 ~ LPC\_GPIO4까지

//총 5개의 값을 가진다. 각 모듈의 설정을 초기화할 때 저장하고 각 모듈이 실행 해야할때마다 로드가

//되도록 한다

//GPIOn의 값을 복사한다

void copy\_GPIOn\_setting(LPC\_GPIO\_TypeDef \*source, LPC\_GPIO\_TypeDef \*dest) ;

//arm processor에 사용되는 LPC\_GPIO에 GPIO\_setting를 로드한다

void load\_LPC\_GPIO\_settingFrom(LPC\_GPIO\_TypeDef GPIO\_setting[5]) ;

//arm processor에 사용되는 LPC\_GPIO를 GPIO\_setting에 저장한다

void save\_LCP\_GPIO\_settingTo(LPC\_GPIO\_TypeDef GPIO\_setting[5]) ;

### EXT\_IO의 종류를 선택하는 함수

Arm 프로세서 kit 에는 총 EXT IO A, B, C 3개를 가지고 이것을 GPIO를 통하여 접근할 수 있다. 이 때GPIO0의 설정을 통하여 하나의 EXT IO를 사용하도록 해야한다. 그렇지 않고 동작하기 위해서는 모듈 외에 다른 EXT IO에 존재하는 모듈에게도 전달하기 때문에 오작동을 일으킬 수 있다.

//EXT\_IO의 종류를 선택하는 함수

void set\_EXT\_IO\_DIRECTION(char EXT\_IO\_NUM) {

if (EXT\_IO\_NUM == 'A' || EXT\_IO\_NUM == 'a') {

LPC\_GPIO0->FIOPIN &= ~(1 << 19); // EXT\_IO\_A On

LPC\_GPIO0->FIOPIN |= (1 << 20); // EXT\_IO\_B Off

LPC\_GPIO0->FIOPIN |= (1 << 21); // EXT\_IO\_C Off

}

else if (EXT\_IO\_NUM == 'B' || EXT\_IO\_NUM == 'b'){

LPC\_GPIO0->FIOPIN |= (1 << 19); // EXT\_IO\_A Off

LPC\_GPIO0->FIOPIN &= ~(1 << 20); // EXT\_IO\_B On

LPC\_GPIO0->FIOPIN |= (1 << 21); // EXT\_IO\_C Off

}

else if (EXT\_IO\_NUM == 'C' || EXT\_IO\_NUM == 'c') {

LPC\_GPIO0->FIOPIN |= (1 << 19); // EXT\_IO\_A Off

LPC\_GPIO0->FIOPIN |= (1 << 20); // EXT\_IO\_B Off

LPC\_GPIO0->FIOPIN &= ~(1 << 21); // EXT\_IO\_C On

}

}

## data part

이 프로그램에서 사용되는 공용변수들을 접근하기 위한 파트로 현재 시스템의 상태, 현재 시간, 입력한 비밀번호, 선택한 락커의 번호, UART통신에 사용되는 플래그 및 카운터의 정보를 접근하게 해준다.

### 시스템 상태 관련 함수

// system state data

//현재의 시스템 모드를 저장하는 변수

int systemMode = SYSTEM\_STATE\_SELECT\_LOCKER\_NUMBER;

//현재의 시스템 모드가 변했는지 저장하는 플래그 변수

bool \_isChangedSystemMode = false;

//현재 시스템 모드가 갱신되었는지 확인하는 질의함수

bool DATA\_isChangedSystemState(void) {

//모드가 변경된이후 다음 변경을 감지하기위해

//\_isChangedSystemMode를 false로 만들고

//true를 반환한다

if (\_isChangedSystemMode == true) {

\_isChangedSystemMode = false;

return true;

}

return false;

}

//systemModed의 setter함수

void DATA\_setSystemState(int mode) {

//만약 이전의 모드와 다르다면 모드가 변경되었으므로

//\_isChangedSystemMode를 true 만든다

if (systemMode != mode) {

systemMode = mode;

\_isChangedSystemMode = true;

}

}

### 현재 시간 갱신하는 함수

//현재 날짜및 시간에서 1초가 지난 후의 시간으로 업데이트한다

void DATA\_timeUpdate(void) {

//각 초, 분, 시, 일, 월, 연 순서로 각각의 진법으로 올림수 나눗셈을 후

//몫은 오버플로우된 올림수로 나머지를 각 단위로 업데이트 하면

//1초 후의 시간으로 업데이트된다

//각 단위의 올림수 이것은 다음 단위에 더 해준다 1초를 업데이트하기위해

//미리 올림수를 1로 시작한다

int carry = 1;

second += carry;

carry = second / 60;

second %= 60;

minute += carry;

carry = minute / 60;

minute %= 60;

hour += carry;

carry = hour / 24;

hour %= 24;

//날(일)의 경우 0일은 존재하지않으므로 처음이 -1

//마지막에 +1하는것으로 위의 연산을 적용할수있다

day--;

day += carry;

//날(일)의 경우 매달마다 수가 달라지므로

//각달의 날수를 저장한 배열의 값을 이용하여 연산을 동일하게 적용한다

carry = day / (daysInMonth[month]);

day %= (daysInMonth[month]);

day++;

//월의 경우도 0월은 존재하지않으므로 처음이 -1

//마지막에 +1하는것으로 위의 연산을 동일하게 적용할수있다

month--;

month += carry;

carry = month / 12;

month %= 12;

month++;

year += carry;

}

### 입력한 비밀번호 관련 함수

//비밀번호에 마지막 한자리 더 추가 하도록 만든다

//4자리가 넘어가는 경우 추가하지않는다

void DATA\_pushbackPassword(int pwdValue) {

if (passwordSize < 4) {

password[passwordSize] = pwdValue;

passwordSize++;

}

}

//비밀번호의 마지막자리를 한자리를 제거한다

//0개 이하일때는 제거하지 않는다

void DATA\_popbackPassword(void) {

if (password > 0) {

passwordSize--;

password[passwordSize] = 0;

}

}

### UART통신에 사용 되는 플래그 및 카운터 관련 함수

// uart data

// 서버와 연결상태에 대한 플래그 변수와 get, set함수

bool \_isServerConnected = false;

bool DATA\_isServerConnected(void) ;

void DATA\_setIsServerConnected(bool flag) ;

//락커를 열기위한 요청이 존재함을 설정하는 변수

//락커 번호와 비밀번호가 입력이 완료되면 true가 되고 요청에 대한 응답이 오면 false로 변경된다

bool \_isRequestedPermission = false;

bool DATA\_isRequestedPermission(void);

void DATA\_setRequsetPermission(bool value) ;

//서버와 연결을 확인을 위한 카운터 변수 및 함수

//메세지가 온다면 갱신이 되고 카운트가 0이 되면 연결이 끊어진 것으로 처리한다

int connectionCounter = TOP\_CONNECTION\_COUNTER;

void DATA\_connectionCountDown(void) ;

bool DATA\_isConnectionCountZero(void);

void DATA\_resetConnectionCounter(void) ;

//요청 메세지의 재전송을 하기위한 카운트변수 및 함수

//카운트 값이 줄어들어 0이되면 재전송을 할 수 있게 만든다

int \_requestSendingMsgCounter = 0;

void DATA\_sendingMsgCountDown(void) ;

void DATA\_resetSendingMsgCounter(void) ;

bool DATA\_isSendingMsgCounterZero(void);

## LCD part

LCD의 화면을 출력하는 부분이다. LCD의 출력속도는 매우 느리므로 갱신이 필요한 경우에만 동작하게 작성하여 프로그램의 반응성을 높였다. LCD\_print를 접두사로 하는 함수들로 데이터에 접근하여 정보를 화면에 출력한다.

### LCD\_print라는 접두사가 붙은 함수

//LCD\_print라는 접두사가 붙은 함수들은 lcd에 화면 출력을 하는 함수들이다.

//화면을 출력 할때 먼저 글자 색과 배경색을 지정한다음 sprintf를 사용하여 출력할 문자열을

//생성하고 GLCD\_displayStringLn로 문자열을 한에 출력하였다.

//비밀번호를 입력하는 화면을 출력

//입력된 비밀번호를 '\*'로 출력하고 입력되니 않은 부분은 공백으로 출력한다

void LCD\_printEnterPasswordPage(void) {

... 생략

//비밀번호의 자리를 표시하기위한 언더바 출력

for (i = 0; i < 4; i++)

GLCD\_displayChar(90 + 40 \* i, 140, '\_');

//입력된 비밀번호의 수많큼 왼쪽에서 '\*'을 출력하고 나머지는 공백을 출력

pwdSize = DATA\_getPasswordSize();

for (i = 0; i < 4; i++) {

if (i < pwdSize)

GLCD\_displayChar(90 + 40 \* i, 130, '\*');

else

GLCD\_displayChar(90 + 40 \* i, 130, ' ');

}

}

//초기화 화면을 출력

void LCD\_printInitPage(void) ;

//락커 번호를 선택하는 화면 출력

void LCD\_printSelectLockerPage(void;

//대기중 화면을 출력

void LCD\_printWaitingPage(void) ;

//열렸을때의 화면을 출력

void LCD\_printOpenLockerPage(void) ;

// 서버와 연결끊어졌을경우 메세지를 출력

void LCD\_printServerDisconnectionMsg(void) ;

//비밀번호를 잘못 입력했을떄 화면을 출력

void LCD\_printWrongPasswordPage(void);

### LCD모듈의 mainTask 함수

//lcd의 io속도가 느리기 때문에 main에서 매번 실행되는 것이 아닌 화면의 표시할 것이 달라졌을 때만

//출력하도록 하였다. 화면상단에 header로 로고, 날짜, 시간을 출력하고 나머지 화면에는

//system mode에 해당하는 화면을 출력한다

void LCD\_mainTask(void) {

//현재 시스템의 상태를 가져온다

int mode = DATA\_getSystemState();

//화면이 갱신될 필요가 없으면 바로 함수를 종료한다

if (isNeedRefreash == false)

return;

//화면의 갱신이 중복되지 않게 플래그 변수를 false로 만든다

isNeedRefreash = false;

//lcd의 헤더를 출력한다

LCD\_printHeader();

//서버와 연결이 끊어졌을 경우 메세지를 출력한다

if (DATA\_isServerConnected() == false) {

LCD\_printServerDisconnectionMsg();

}

//서버와 연결이 끊어졌다가 연결됫을 경우 메세지를 지운다

else {

GLCD\_clearLn(Line9);

}

//만약 시스템 모드가 변경된다면 header를 제외한 부분이 다른 부분을 출력하기 위해 이전 화면을 지운다

if (DATA\_isChangedSystemState() == true) {

LCD\_clearPage();

}

//각 모드에 맞는 lcd 화면을 출력한다

switch (mode) {

//락커 번호를 선택하는 상태

case SYSTEM\_STATE\_SELECT\_LOCKER\_NUMBER: LCD\_printSelectLockerPage(); break;

//비밀번호를 입력받는 상태

case SYSTEM\_STATE\_ENTER\_PASSWORD: LCD\_printEnterPasswordPage(); break;

//락커 열기 성공 상태

case SYSTEM\_STATE\_OPEN\_SUCCESS: LCD\_printOpenLockerPage(); break;

//락커 열기 실패 상태

case SYSTEM\_STATE\_OPEN\_FAIL: LCD\_printWrongPasswordPage(); break;

//락커 열기 요청에 대한 대기 상태

case SYSTEM\_STATE\_WAITING\_RESPONSE: LCD\_printWaitingPage(); break;

default:break;

}

}

## Keypad part

사용자로부터 락커 번호 및 비밀번호, 입력 받은 값 삭제, 이전/다음페이지 이동 및 락커 닫기를 입력을 처리한다.

### keypad의 mainTask함수

//이전의 눌린 키패드의 값

//키패드의 값이 변할 때마다 갱신 되어야한다

int prevKeypadValue = KEYPAD\_UP;

//system mode에 따라 서로 다른 동작을 한다

//락커 번호와 비밀번호를 입력 및 정정, 뒤로 가기, 다음 화면, 닫기 버튼 등을 처리한다

void Keypad\_mainTask(void) {

int rawKeypadValue = 0;

char mappedValue = '0';

int lockerNumber = 0;

int mode = DATA\_getSystemState();

//keypad를 작동시키 위해 GPIO 세팅을 로드 한다

load\_LPC\_GPIO\_settingFrom(keypad\_GPIO\_SETTING);

//keypad가 눌린 값을 얻고 매핑된 키패드의 값을 구한다

rawKeypadValue = KEYPAD\_getPressKeyNumber(DEFAULT\_KEYPAD\_EXIO\_NUM);

mappedValue = keypadMappingVector[rawKeypadValue];

//눌린것이 없다면 이전 값을 갱신하고 반환한다

if (rawKeypadValue == KEYPAD\_UP) {

prevKeypadValue = KEYPAD\_UP;

return;

}

//키패드가 down이 되는 상태 일때만 한번만 입력이 되도록 하여 키패드가 눌린 이후

//중복되게 입력됨을 방지한다

if (rawKeypadValue != KEYPAD\_UP && prevKeypadValue == KEYPAD\_UP) {

prevKeypadValue = rawKeypadValue;

//서버와 연결이 끊어졌을 경우 입력을 처리하지 않는다

if (DATA\_isServerConnected() == false) {

return ;

}

//각 모드에 따른 keypad 동작을 처리한다

//각 모드에 대해 올바른 입력을 했을때 LCD\_setRefresh를

//호출하는 것으로 lcd 화면을 갱신한다

switch (mode){

//락커의 번호를 선택하는 모드

case SYSTEM\_STATE\_SELECT\_LOCKER\_NUMBER: {

//번호를 눌렀을 때의 처리

//락커 번호는 1~18 까지 있으므로 범위를 넘어가는 값은

//처리하지않다. 범위를 안에 있다면 선택한 락커 번호를 갱신한다.

if (mappedValue >= '0' && mappedValue <= '9') {

lockerNumber = DATA\_getSelectedLockerNumber();

lockerNumber = lockerNumber \* 10 + mappedValue - '0';

//선택한 락커번호를 갱신한다

if (lockerNumber <= 18) {

LCD\_setRefresh();

DATA\_setSelectedLockerNumber(lockerNumber);

}

}

//입력된 마지막번호를 지우는 동작을 처리한다

else if (mappedValue == KEY\_CLEAR) {

LCD\_setRefresh();

lockerNumber = DATA\_getSelectedLockerNumber();

lockerNumber /= 10;

DATA\_setSelectedLockerNumber(lockerNumber);

}

//다음페이지를 선택 동작에 대한 처리

else if (mappedValue == KEY\_GO\_NEXT) {

lockerNumber = DATA\_getSelectedLockerNumber();

//선택한 락커의 번호가 유효할때만 다음화면으로 넘어간다

if (lockerNumber <= 18 && lockerNumber >= 1) {

LCD\_setRefresh();

DATA\_setSystemState(SYSTEM\_STATE\_ENTER\_PASSWORD);

}

}

break;

}

//선택한 락커에 대한 비밀번호를 입력하는 모드

case SYSTEM\_STATE\_ENTER\_PASSWORD: {

//번호 입력 시 비밀번호 입력 처리

if (mappedValue >= '0' && mappedValue <= '9') {

LCD\_setRefresh();

DATA\_pushbackPassword(mappedValue - '0');

}

//입력된 마지막 번호 삭제 처리

else if (mappedValue == KEY\_CLEAR) {

LCD\_setRefresh();

DATA\_popbackPassword();

}

//다음페이지를 처리한다

else if (mappedValue == KEY\_GO\_NEXT) {

LCD\_setRefresh();

DATA\_setRequsetPermission(true);

DATA\_setSystemState(SYSTEM\_STATE\_WAITING\_RESPONSE);

}

//비밀번호 입력 중 뒤로 돌아가기 처리

else if (mappedValue == KEY\_GO\_BACK) {

//입력된 비밀번호와 락커번호를 삭제한다

DATA\_clearPassword();

DATA\_setSelectedLockerNumber(LOCKER\_NUMBER\_UNSELECTED);

LCD\_setRefresh();

DATA\_setSystemState(SYSTEM\_STATE\_SELECT\_LOCKER\_NUMBER);

}

break;

}

//서버와 통신을 하는 모드 통신 중에는 입력 받지않는다

case SYSTEM\_STATE\_WAITING\_RESPONSE: {

break;

}

//락커가 열렸을 때

case SYSTEM\_STATE\_OPEN\_SUCCESS: {

//락커 닫기를 눌렀을 때 락커를 닫고 비밀번호와 선택한

//락커 번호를 초기화하고 처음 화면으로 돌아간다

if (mappedValue == KEY\_CLOSE) {

MOTOR\_setCloseLocker();

LCD\_setRefresh();

DATA\_setSystemState(SYSTEM\_STATE\_SELECT\_LOCKER\_NUMBER);

DATA\_clearPassword();

DATA\_setSelectedLockerNumber(LOCKER\_NUMBER\_UNSELECTED);

}

break;

}

//비밀번호에 틀려 열기를 실패 했을 때

case SYSTEM\_STATE\_OPEN\_FAIL: {

//돌아기기 또는 다음화면 버튼을 누르면 비밀번호와 선택한

//락커 번호를 초기화하고 처음 화면으로 돌아간다

if (mappedValue == KEY\_GO\_BACK || mappedValue == KEY\_GO\_NEXT) {

LCD\_setRefresh();

DATA\_setSystemState(SYSTEM\_STATE\_SELECT\_LOCKER\_NUMBER);

DATA\_clearPassword();

DATA\_setSelectedLockerNumber(LOCKER\_NUMBER\_UNSELECTED);

}

break;

}

default:

break;

}

}

}

## MOTOR part

모터를 움직여 락커를 열림과 닫힘을 구현한 파트이다. 모터는 스텝모터를 사용하였고 반바퀴를 회전하는 것으로 락커의 열림과 닫힘을 구현하였다.

### 모터 상태, 설정 변수 및 함수

bool isNeedLockerMotorMove = false;

//모터를 움직어 락커를 열도록 모터 상태를 세팅한다

void MOTOR\_setOpenLocker() {

//이전의 락커상태와 다를때만 모터를 움직이도록 세팅한다

if (LockerState != LOCKER\_OPEN)

isNeedLockerMotorMove = true;

LockerState = LOCKER\_OPEN;

}

//모터를 움직여 락커를 닫도록 모터 상태를 세팅한다

void MOTOR\_setCloseLocker() {

//이전의 락커상태와 다를때만 모터를 움직이도록 한다

if (LockerState != LOCKER\_CLOSE)

isNeedLockerMotorMove = true;

LockerState = LOCKER\_CLOSE;

}

### 스탭 모터 작동함수

//스텝모터는 내부에 2개의 전자석 a, b를 가진다. 전자석 a, b의 전류방향을 각각 a, na와 b, bn으로 설정할

//있다. 이때 두개의 전자석의 전류방향을 선택하는 것이 하나의 스텝이며 이러한 스텝의 시퀸스를 거치는

//것으로 하나의 스텝 사이클을 구성한다. 또한 이런 12번의 스텝사이클을 반복하는 것으로 모터를

//한바퀴 움직이게 된다. 따라서 반바퀴만 움직이려면 12스텝사이클의 반인 6번의 스텝사이클을 돌면 된다.

//또한 하나의 스텝을 할떄 마다 모터가 움직이는 시간을 주기위해 일정시간만큼의 딜레이가 필요하다.

//motor를 반시계방향으로 반바퀴가 움직이도록 만든다

void MOTOR\_moveCCWHalfCycle() {

int count = 0;

for (count = 0; count < 6; count++) {

//a, b 방향으로 전류를 흘린다

GPIO\_SetValue(GPIO\_PORT\_0, GPIO\_PIN\_5);

GPIO\_ClearValue(GPIO\_PORT\_0, GPIO\_PIN\_10);

GPIO\_SetValue(GPIO\_PORT\_0, GPIO\_PIN\_23);

GPIO\_ClearValue(GPIO\_PORT\_0, GPIO\_PIN\_24);

//모터가 움직일 시간을 주기위한 딜레이

Delay(MOTOR\_DELAY\_TIME);

//na, b 방향으로 전류를 흘린다

GPIO\_ClearValue(GPIO\_PORT\_0, GPIO\_PIN\_5);

GPIO\_SetValue(GPIO\_PORT\_0, GPIO\_PIN\_10);

GPIO\_SetValue(GPIO\_PORT\_0, GPIO\_PIN\_23);

GPIO\_ClearValue(GPIO\_PORT\_0, GPIO\_PIN\_24);

//모터가 움직일 시간을 주기위한 딜레이

Delay(MOTOR\_DELAY\_TIME);

//na, nb 방향으로 전류를 흘린다

GPIO\_ClearValue(GPIO\_PORT\_0, GPIO\_PIN\_5);

GPIO\_SetValue(GPIO\_PORT\_0, GPIO\_PIN\_10);

GPIO\_ClearValue(GPIO\_PORT\_0, GPIO\_PIN\_23);

GPIO\_SetValue(GPIO\_PORT\_0, GPIO\_PIN\_24);

//모터가 움직일 시간을 주기위한 딜레이

Delay(MOTOR\_DELAY\_TIME);

//a, nb 방향으로 전류를 흘린다

GPIO\_SetValue(GPIO\_PORT\_0, GPIO\_PIN\_5);

GPIO\_ClearValue(GPIO\_PORT\_0, GPIO\_PIN\_10);

GPIO\_ClearValue(GPIO\_PORT\_0, GPIO\_PIN\_23);

GPIO\_SetValue(GPIO\_PORT\_0, GPIO\_PIN\_24);

//모터가 움직일 시간을 주기위한 딜레이

Delay(MOTOR\_DELAY\_TIME);

}

}

### 모터의 mainTask 함수

void MOTOR\_mainTask() {

//모터가 동작할 일이 없다면 바로 종료한다

if (isNeedLockerMotorMove == false)

return;

//중복으로 모터가 동작을 방지를 위해 플래그 변수를 false로 변경한다

isNeedLockerMotorMove = false;

//모터의 GPIO의 값을 로드한다

load\_LPC\_GPIO\_settingFrom(motor\_GPIO\_SETTING);

//Locker의 상태에 따라 모터를 움직여 락커를 열거나 닫는다

if (LockerState == LOCKER\_OPEN) {

MOTOR\_moveCCWHalfCycle();

}

else if (LockerState == LOCKER\_CLOSE) {

MOTOR\_moveCCWHalfCycle();

}

}

## UART part

UART통신을 이용하여 서버와 연결하여 시간을 동기화한다. 이후 메시지를 수신해서 수신 받은 메시지에 대해 처리한다. UART통신에서는 사용되는 메시지들은 미리 정해 놓은 메시지 마크를 조합 하여 사용한다. Polling 방식으로 전송 및 수신을 한다. 이 때 전송되는 케이블의 버퍼를 사용하여 단순 전송을 하고, 수신 받을 때는 수신 받은 1-byte의 데이터마다 echo를 전송하여 전송 측에 전송에 성공함을 알리는 것으로 통신을 구현하였다.

### UART 통신을 위한 프로토콜에 사용되는 mark

// uart msg mark

// uart 통신을 위한 프로토콜에 사용되는 mark

// mark들을 조합하여 통신에 필요한 메시지를 만든다

// protocol format : "HEADMARK + BODYMARK or data + FOOTMARK"

#define UART\_MSG\_HEADMARK\_handshake\_send "send\_handshake"

#define UART\_MSG\_HEADMARK\_handshake\_receive "receive\_handshake"

#define UART\_MSG\_HEADMARK\_connected\_confirm "connected\_confirm"

#define UART\_MSG\_HEADMARK\_send "send"

#define UART\_MSG\_HEADMARK\_receive "receive"

#define UART\_MSG\_HEADMARK\_disconnect "disconnect"

#define UART\_MSG\_HEADMARK\_SyncTime "Syc\_time"

#define UART\_MSG\_HEADMARK\_request\_permission "request\_permission"

#define UART\_MSG\_HEADMARK\_response\_permission "reseponse\_permission"

#define UART\_MSG\_HEADMARK\_check\_connection "check\_connection"

#define UART\_MSG\_BODYMARK\_yes "yes"

#define UART\_MSG\_BODYMARK\_no "no"

#define UART\_MSG\_BODYMARK\_success "success"

#define UART\_MSG\_FOOTMARK\_end "end\n"

### UART의 연결상태를 나타내는 enum 선언

typedef enum { UART\_CONNECTED, UART\_DISCONNECTED } UART\_CONNECTION\_STATE;

//UART의 연결상태를 저장하는 변수

UART\_CONNECTION\_STATE UART\_currentState = UART\_DISCONNECTED;

### 수신받은 버퍼 관련함수

uint8\_t receivedBuffer[UART\_BUFFER\_SIZE]; // 문자열 저장 버퍼

uint8\_t bufferCount = 0; //버퍼에 Write할 위치

void UART\_clearReceivedBuffer(void) ; //전송받은 버퍼를 초기화 한다

### UART 전송 및 수신 함수

//polling 방식으로 uart0로 메세지를 전송한다.

//rs323 -> usb 로 가는 방향에 버퍼가 존재하여 메세지를 단순히 폴링 방식으로 보내주어도 된다.

//하지만 이경우 수신 측에서 버퍼에 오버 플로우가 일어나지 안도록 폴링 방식으로 수신 받아야한다.

void UART\_sendMsg(char msg[UART\_BUFFER\_SIZE]) {

UARTPuts(LPC\_UART0, msg);

}

//수신 받는 경우 또한 폴리 방식으로 수신 받은 한바이트에 대하여 echo를 전송하여 성공적으로 수신

//받음을 알려주어서 전송 측이 다음 바이트를 전송하는 것을 반복하여 msg를 수신한다.

//수신에 성공하면 수신 버퍼에 메세지를 저장하고 true를 반환하고

//수신 받은 것이 없거나, 버퍼가 가득 찬 경우 false를 반환한다.

bool UART\_receiveMsg(char buffer[UART\_BUFFER\_SIZE]) {

//시용되어질 변수들 초기화

bool isReceivedMsg = false;

int i = 0;

char received\_char;

bool isReceivedByte = false;

char tempReceivedMsg[UART\_BUFFER\_SIZE];

//버퍼가 가득차면 수신 받을 수 없으므로 종료한다

if (bufferCount == UART\_BUFFER\_SIZE) return false;

//현재 버퍼에서 최대로 수신 받을 수 있는 만큼만 수신을 검사한다

for (i = bufferCount; i < UART\_BUFFER\_SIZE; i++) {

//polling 방식으로 byte를 수신한다

isReceivedByte = UART\_Receive(LPC\_UART0,

(uint8\_t \*)tempReceivedMsg, UART\_BUFFER\_SIZE, NONE\_BLOCKING);

//수신 받은 것이 없을 경우 생략한다

if (isReceivedByte == false)

continue;

//수신 받은 byte를 가져온다

received\_char = tempReceivedMsg[0];

//수신 받은 byte에 대하여 echo를 전송한다

UARTPuts(LPC\_UART0, &received\_char);

//수신 받은 byte를 버퍼에 저장한다

buffer[bufferCount] = received\_char;

bufferCount++;

//수신 받은 byte가 개행문자일 경우 메세지의 끝이므로

//수신을 성공하였다

if (received\_char == '\n') {

isReceivedMsg = true;

break;

}

}

//수신 받음의 플래그 변수를 반환한다

return isReceivedMsg;

}

### UART\_make를 접두사로 가지는 함수

//UART\_make를 접두사로 가지는 함수들은 모두 msg mark를 조합하여

//보낼 메세지를 만든다

//handshake msg를 만든다

void UART\_makeHandShakeMsg(char msg[UART\_BUFFER\_SIZE]) ;

//check connection msg를 만든다

void UART\_makeCheckConnectionMsg(char msg[UART\_BUFFER\_SIZE]);

//discconection msg를 만든다

void UART\_makeDisconnectionMsg(char msg[UART\_BUFFER\_SIZE]) ;

//락커를 열기를 위한 권한을 요청하는 msg를 만든다

void UART\_makeRequestPermssionMsg(char msg[UART\_BUFFER\_SIZE]) ;

### UART\_is를 접두사로 가지는 함수

//uart\_is를 접두사로 가지는 함수들은 msg를 검사하는 함수들이다.

//msg mark를 이용하여 올바른 msg를 생성하고 인자로 들어온 msg를 검사하여 해당하는 msg인지 검사한다

//msg가 end mark로 끝나는지 검사한다.

//이 함수로 msg가 완전히 끝나는지 검사하는 역할을 한다.

//공백을 구분으로 토큰으로 나누어 마지막 토큰이 msg end mark인것을 검사한다.

bool UART\_isMsgMeetEndMark(char msg[UART\_BUFFER\_SIZE]) {

//strtok로 msg 넣는다면 msg의 내용이 변경되므로 임시 버퍼로 복사하여 검사한다.

char tempBuffer[UART\_BUFFER\_SIZE];

//end mark를 저장할 변수

char endMark[10];

//token의 시작 포인터를 저장하는 변수

char \*tokenPtr = NULL;

//문자열들을 초기화한다

memset(endMark, 0x00, sizeof(endMark));

memset(tempBuffer, 0x00, sizeof(tempBuffer));

sprintf(endMark, "%s", UART\_MSG\_FOOTMARK\_end);

strcpy(tempBuffer, msg);

//임시 버퍼를 ' '(공백)으로 토큰을 나누고 token을 검사한다.

tokenPtr = strtok(tempBuffer, " ");

while (tokenPtr != NULL) {

if (strcmp(tokenPtr, endMark) == 0) return true;

tokenPtr = strtok(NULL, " ");

}

return false;

}

//hand shake msg 인지 검사한다.

//handShake msg format :

//"UART\_MSG\_MARK\_disconnect UART\_MSG\_MARK\_end"

bool UART\_ishandShakeMsg(char msg[UART\_BUFFER\_SIZE]) ;

//disconection msg 인지 검사한다.

//disconection msg format :

//"UART\_MSG\_MARK\_disconnect UART\_MSG\_MARK\_end"

bool UART\_isDisconnectMsg(char msg[UART\_BUFFER\_SIZE]) ;

//시간을 동기화하는 msg 인지 검사한다.

// SycTime Format msg :

//"UART\_MSG\_MARK\_SyncTime yyyyMMddhhmmss UART\_MSG\_MARK\_end"

bool UART\_isTimeSyncMsg(char msg[UART\_BUFFER\_SIZE]);

//check connection msg 인지 검사

//check connection msg format :

//"UART\_MSG\_MARK\_check\_connection UART\_MSG\_MARK\_end"

bool Uart\_isCheckConnectionMsg(char msg[UART\_BUFFER\_SIZE]);

//check connection msg 인지 검사

//check connection msg format :

//"UART\_MSG\_MARK\_check\_connection UART\_MSG\_MARK\_end"

bool UART\_isReseponsePermissionMsg(char msg[UART\_BUFFER\_SIZE]);

//ResponsePemissionOK msg 인지 검사

//ResponsePemissionOK msg format :

//"UART\_MSG\_MARK\_check\_connection UART\_MSG\_MARK\_end"

bool UART\_isResponsePemissionOK(char msg[UART\_BUFFER\_SIZE]) ;

### UART로 수신 받은 시간 동기화 함수

void UART\_syncTime(char raw\_msg[UART\_BUFFER\_SIZE]) {

... 생략

//받은 메세지중 시간부분의 문자열을 추출한다

memset(strTime, 0x00, sizeof(strTime));

for (i = 0; i < 14; i++) {

strTime[i] = raw\_msg[9 + i] - '0';

}

//추출된 문자열을 정수로 변환한다

year = strTime[0] \* 1000 + strTime[1] \* 100 + strTime[2] \* 10 + strTime[3];

month = strTime[4] \* 10 + strTime[5];

day = strTime[6] \* 10 + strTime[7];

hour = strTime[8] \* 10 + strTime[9];

minute = strTime[10] \* 10 + strTime[11];

second = strTime[12] \* 10 + strTime[13];

//정수로 변환된 시간및 날짜를 설정한다

DATA\_setTime(year, month, day, hour, minute, second);

}

### UART의 mainTask함수

//서버와 연결, 메세지 전송 및 수신을 처리한다

void UART\_mainTask(void) {

//보낼 메세지를 위한 버퍼 및 초기화

char sendingBuffer[UART\_BUFFER\_SIZE];

memset(sendingBuffer, 0, sizeof(sendingBuffer));

//uart의 상태가 UART\_DISCONNECTED일때

if (UART\_currentState == UART\_DISCONNECTED) {

DATA\_setIsServerConnected(false);

UART\_receiveMsg((char \*)receivedBuffer);

//수신 받은 메세지가 완전한 format을 이룰때

if (UART\_isMsgMeetEndMark((char \*)receivedBuffer) == true) {

//hand shake msg 수신시 msg에 대한 응답을 하고 uart 상태를 변경한다

if (UART\_ishandShakeMsg((char \*)receivedBuffer)) {

UART\_makeHandShakeMsg((char\*) sendingBuffer);

UART\_sendMsg(sendingBuffer);

UART\_currentState = UART\_CONNECTED;

DATA\_setIsServerConnected(true);

}

//disconection msg 수신시 msg에 대한 응답을 하고 uart 상태를 변경한다

else if (UART\_isDisconnectMsg((char \*)receivedBuffer)) {

UART\_makeDisconnectionMsg(sendingBuffer);

UART\_sendMsg(sendingBuffer);

UART\_currentState = UART\_DISCONNECTED;

DATA\_setIsServerConnected(false);

DATA\_resetConnectionCounter();

}

//버프를 비워서 다음 msg를 받을 수 있도록 만든다

UART\_clearReceivedBuffer();

}

}

//uart 연결 중일때

else if (UART\_currentState == UART\_CONNECTED) {

//먼저 msg를 수신 받는다

UART\_receiveMsg((char \*)receivedBuffer);

//수신받은 메세지가 완전한 format을 이룰 때 msg를 처리한후 receivedBuffer비워

//다음 메세지를 수신받을 수 있도록 만든다

if (UART\_isMsgMeetEndMark((char\*)receivedBuffer) == true) {

//disconection msg 수신시 msg에 대한 응답을 하고

//uart 상태를 변경한다

if (UART\_isDisconnectMsg((char \*)receivedBuffer)) {

UART\_makeDisconnectionMsg(sendingBuffer);

UART\_sendMsg(sendingBuffer);

UART\_currentState = UART\_DISCONNECTED;

DATA\_setIsServerConnected(false);

DATA\_resetConnectionCounter();

}

//time sync msg 수신시 msg에 대한 응답을 하고 시간을 동기화한다

if (UART\_isTimeSyncMsg((char\*)receivedBuffer) == true) {

UART\_syncTime((char\*)receivedBuffer);

DATA\_resetConnectionCounter();

}

if (UART\_isReseponsePermissionMsg((char \*)receivedBuffer) == true) {

//비밀번호가 맞으면 시스템 상태를 변경하고 락커 열도록 설정한다

if (UART\_isResponsePemissionOK((char \*)receivedBuffer) == true) {

DATA\_setSystemState(SYSTEM\_STATE\_OPEN\_SUCCESS);

MOTOR\_setOpenLocker();

}

//틀린경우 시스템 상태를 변경한다

else {

DATA\_setSystemState(SYSTEM\_STATE\_OPEN\_FAIL);

}

//lcd 화면을 갱신한다

LCD\_setRefresh();

//요청에 대한 응답이 왔으므로 더이상 메시지를 보내지 않기위해

//요청 메세지가 없음으로 변경한다

DATA\_setRequsetPermission(false);

//성공적인 메세지 교환이므로 연결위한 카운터를 초기화한다

DATA\_resetConnectionCounter();

}

//받은 메세지가 서버와 연결을 확인하는 메세지일 경우

if (Uart\_isCheckConnectionMsg((char \*)receivedBuffer) == true) {

//메세지를 조합하여 전송한다

UART\_makeCheckConnectionMsg((char \*)sendingBuffer);

UART\_sendMsg(sendingBuffer);

//서버와 연결상태를 갱신한다

DATA\_setIsServerConnected(true);

DATA\_resetConnectionCounter();

}

//버프를 비워서 다음 msg를 받을 수 있도록 만든다

UART\_clearReceivedBuffer();

}

//전송할 락커를 열기위해 보낼 메세지가 있을 경우

if (DATA\_isRequestedPermission() == true) {

//메세지 전송하고 일정시간동안 응답이 없으면 응답이 올때까지 재전송을 한다.

if (DATA\_isSendingMsgCounterZero() == true) {

//메세지를 조합하여 전송한다

UART\_makeRequestPermssionMsg((char\*)sendingBuffer);

UART\_sendMsg((char \*)sendingBuffer);

//화면을 갱신하도록 설정한다

LCD\_setRefresh();

//메세지의 재전송 카운터를 초기화한다

DATA\_resetSendingMsgCounter();

}

}

//일정시간동안 메세지를 받지 못한다면 연결이 끊어진 것으로 처리하기 위해 시스템상태를

//DISCONNECTED로 변경한다

if (DATA\_isConnectionCountZero()) {

DATA\_resetConnectionCounter();

UART\_currentState = UART\_DISCONNECTED;

}

}

return;

}

## Python arm kit server part

ARM kit와 UART 통신을 처리하는 서버 파트이다. 시리얼 통신으로 ARM kit와 데이터를 주고받는다. 이 때 polling 방식으로 통신을 하므로, 매 통신이후 sleep을 하여 CPU의 부하를 줄였다. ARM kit에서 락커의 열기 인증 메시지를 제외한 기본적인 통신 기능들은 Python ARM kit server에서 전송하고 ARM kit가 응답하는 방식으로 통신을 한다.

### UART를 이용하여 메시지를 전송 및 수신하는 함수

def receiveMsg(self):

return str(self.serialConnectionObject.readline())[2:-1]

#메시지를 byte 로 변환한후 한글자 전송하고 echo를 받는 것을 반복하여 메시지 스트림을 보낸다

def sendMsg(self, msg, showEcho=false):

for c in msg:

bChar = bytes(c, encoding = self.MSG\_ENCONDDING\_TYPE)

self.serialConnectionObject.write(bChar)

echoChar = ""

while (1):

echoChar = self.receiveMsg()

if (echoChar != ''):

break

if (showEcho):

print("send : %s echo : %s" % (bChar, echoChar))

return

### Python arm kit server의 기본적인 통신기능 함수

통신에 필요한 handshaking, check connection, disconnection기능을 수행한다.

# handshaking 수행 하는 함수

def makeHandShake(self, showLog=false, timeOut=HAND\_SHAKE\_TIME\_LIMIT):

curtime = time.clock

startTime = curtime()

handShakeMsg = "send\_handshake end\n"

# handshaking 메시지를 전송한다

self.sendMsg(handShakeMsg)

msg = ""

while 1:

#polling 방식의 통신의 cpu 부하를 줄이기 위한 sleep

self.sleep()

rcvMsg = self.receiveMsg()

#응답 시간이 초과된다면 재전송한다

if curtime() - startTime > 1:

self.sendMsg(handShakeMsg)

break

msg += rcvMsg

msgTokenList = msg.split()

#응답메시지가 완전히 전송히 끝날떄까지 대기하도록 만든다

if len(msgTokenList) <2 or msgTokenList[-1] != "end\\n":

continue

if msgTokenList[0] == "receive\_handshake" and msgTokenList[1] == 'end\\n': msg = "receive\_handshake end\n"

return true

return false

#disconnection 수행하는 함수

def disconnectToArmKit(self, showLog=false) :

curtime = time.clock

starttime = curtime()

dcsMsg = "disconnect end\n"

dcrMsg = "disconnect success end\\n"

#disconection 메시지를 전송한다

self.sendMsg(dcsMsg)

msg = ""

while true:

#polling 방식의 통신의 cpu 부하를 줄이기 위한 sleep

self.sleep()

#응답 시간이 초과된다면 재전송한다

if curtime() - starttime > 3:

self.sendMsg(dcsMsg)

s = self.receiveMsg()

#수신받은 것이 없다면 대기한다

if s == "": continue

msg += s

#지금까지 받은 버퍼에 응답메세지가 존재하는지 검사한다

if dcrMsg in msg:

break

if len(msg) > self.MAX\_MSG\_LEN :

self.sendMsg(dcsMsg)

return true

# check connection을 수행하는 함수

def CheckConnection(self, showLog = false):

#check connection 메시지를 전송한다

self.sendMsg("check\_connection end\n")

msg = ""

startTime = time.time();

timeLimit = 5

rcvmsg = "check\_connection end\\n"

while 1:

#응답시간이 만료되었다면 연결이 끊어진것으로 처리한다

if time.time() - startTime > timeLimit:

return false

msg += self.receiveMsg()

if rcvmsg in msg :

return true

return false

### ARM kit로부터 락커 열기 요청에 대한 응답을 처리하는 함수

해당하는 락커 번호의 비밀번호와 비교하여 응답메시지를 전송한다. 비밀번호가 일치하는 경우 락커의 비밀번호를 다른 랜덤한 4자리 수로 변경한다

def responsePermissionMsg(self, msg, showLog = false):

tokenList = msg.split()

pwd = tokenList[1]

lockerNum = tokenList[2]

#선택한 락커의 번호와 비밀번호가 일치하면 비밀번호를 갱신하고 인증성공으로 처리한다

if lockerNum in self.lockerDB and self.lockerDB[lockerNum]["lockerPassword"] == pwd:

self.lockerDB[lockerNum]["lockerPassword"] = "%04d"%(random.randint(0,9999))

result = true

else :

result = false

#인증 결과를 ARM kit로 전송한다

if result == true:

msg ="reseponse\_permission %s end\n"%("yes")

else:

msg ="reseponse\_permission %s end\n"%("no")

if showLog:

print("server send : %s"%( msg))

self.sendMsg(msg)

### ARM kit와 통신연결을 하는 함수

ARM kit와 UART 통신연결을 처리하는 함수이다. Python ARM kit server와 ARM kit가 연결된 후 서버 측이 중지 되었다가 재실행되는 경우가 존재하므로 ARM kit의 통신 상태를 리셋 하기 위해 disconnect를 가장 먼저 실행한다. 그 다음 hand shake를 통해 연결을 맺고 키트와 서버의 시간을 동기화한다.

def connectToArmKit(self, showLowg = false):

self.disconnectToArmKit()

self.printTimeLog("disconnection success")

self.printTimeLog("start hand shake")

while(1):

if self.makeHandShake() == true:

break

self.printTimeLog("retry hand shake")

self.printTimeLog("hand shake success")

self.sendSynTimeMsg()

self.printTimeLog("sync Time complete")

return

### Python ARM kit server의 main Task

시리얼포트와 연결을 하고, ARM 키트와 hand shake를 맺는 것으로 연결을 확인한다. 그리고 Python ARM kit server와 ARM kit의 시간을 동기화한다. 또한 연결을 확인하는 메시지를 지속적으로 보내 ARM kit와 연결을 유지하고 락커 열기 인증 요청에 응답한다.

def main(self, \_lockerDB = {}):

#lockerDB를 초기화한다

self.setLockerDB(\_lockerDB)

#uart 연결을 한다

self.connectSerial(self.serialConnectionSetting)

if self.serialConnectionObject == None:

self.printTimeLog("can not make serial connection")

exit()

#ArmKit와 연결을 시도한다

self.connectToArmKit()

checkConnectionTime = self.CHECK\_CONNECTION\_INTERVAL

msgBuffer = ""

while true:

#polling 부하를 줄이기 위한 sleep

self.sleep()

#Arm kit와 연결을 확인하기위해 일정시간마다 연결메세지를 전송한다

checkConnectionTime -= 1

if checkConnectionTime <= 0:

checkConnectionTime = self.CHECK\_CONNECTION\_INTERVAL

isOk = self.CheckConnection()

#ARM kit와 연결이 끊어지면 다시 연결한다

if isOk:

self.printTimeLog("check ok")

else :

self.connectToArmKit()

#수신받은 내용을 버퍼에 추가한다

curReceivedData = self.receiveMsg()

msgBuffer += curReceivedData

#수신받은 내용이 존재하지않으면 무시한다

if curReceivedData != "" :

#완전한 메세지 포멧이 아니라면 무시한다

tokenList = msgBuffer.split()

if tokenList[-1] != 'end\\n':

continue;

self.printTimeLog("received : %s"%(msgBuffer))

#받은 메세지가 인증요청 메세지의 경우에대한 처리

if self.isPemissionMsg(msgBuffer) != false:

self.responsePermissionMsg(msgBuffer)

msgBuffer = ""

continue

self.disconnectToArmKit()

## Flask web Server part

Flask 웹서버 파트이다. 웹에서 사용자의 락커 생성 및 반납 과 비밀번호 확인할 수 있는 페이지를 제공한다. Python ARM kit server와 Flask web Server가 동시에 실행 되어야하기 때문에 두 개의 스레드에서 실행되도록 하였다.

### 사용자의 락커 신청 처리 함수

##사용자가 요청한 락커 사용 신청을 처리한다

##락커의 비밀번호를 랜덤으로 설정하고 락커를 등록한다

@app.route('/checkMakeNewLocker', methods = ['post'] )

def checkMakeNewLocker():

userId = request.form['id']

lockerNumber = request.form['lockerNumber']

#락커의 비밀번호를 랜덤으로 생성하고 신청한 아이디에 선택한 락커를 배정한다

lockerPassword = "%04d"%(random.randint(0,9999))

lockerDB[str(lockerNumber)] = { "userID" : userId , "lockerPassword" : lockerPassword }

return redirect(url\_for("completeMakeNewLockerPage", userId = userId))

### 사용자의 락커 반납 처리 함수

##사용한 락커를 반납한다

@app.route('/returnLocker', methods = ['POST', 'GET'])

def returnLocker():

userId = request.form["userId"]

lockerNumber = request.form["lockerNumber"]

#해당 락커를 반납시킨다

lockerDB[ lockerNumber ] = None

if userId == "root":

return redirect(url\_for('rootMainPage', id = userId))

else :

return redirect(url\_for('userMainPage', id = userId))

### Python ARM kit server와 Flask web Server의 스레드 생성 및 실행

##flask 웹서버와 python 서버를 동시에 실행하기위해 두개의 스레드로 만들어 실행한다

if \_\_name\_\_ == "\_\_main\_\_":

initLockerDb()

try:

threadList = []

armServer = arm\_m3\_kit\_server.arm\_m3\_kit\_server()

#Python ARM kit server의 스레드를 생성하고 실행한다

thArmServer = threading.Thread(target = armServer.main, kwargs={'\_lockerDB': lockerDB} )

thArmServer.start()

#Flask web Server의 스레드를 생성하고 실행한다

thFlaskServer = threading.Thread(target = app.run, args = ('192.168.0.2', 8080) )

thFlaskServer.start()

threadList +=[thArmServer, thFlaskServer]

for t in threadList:

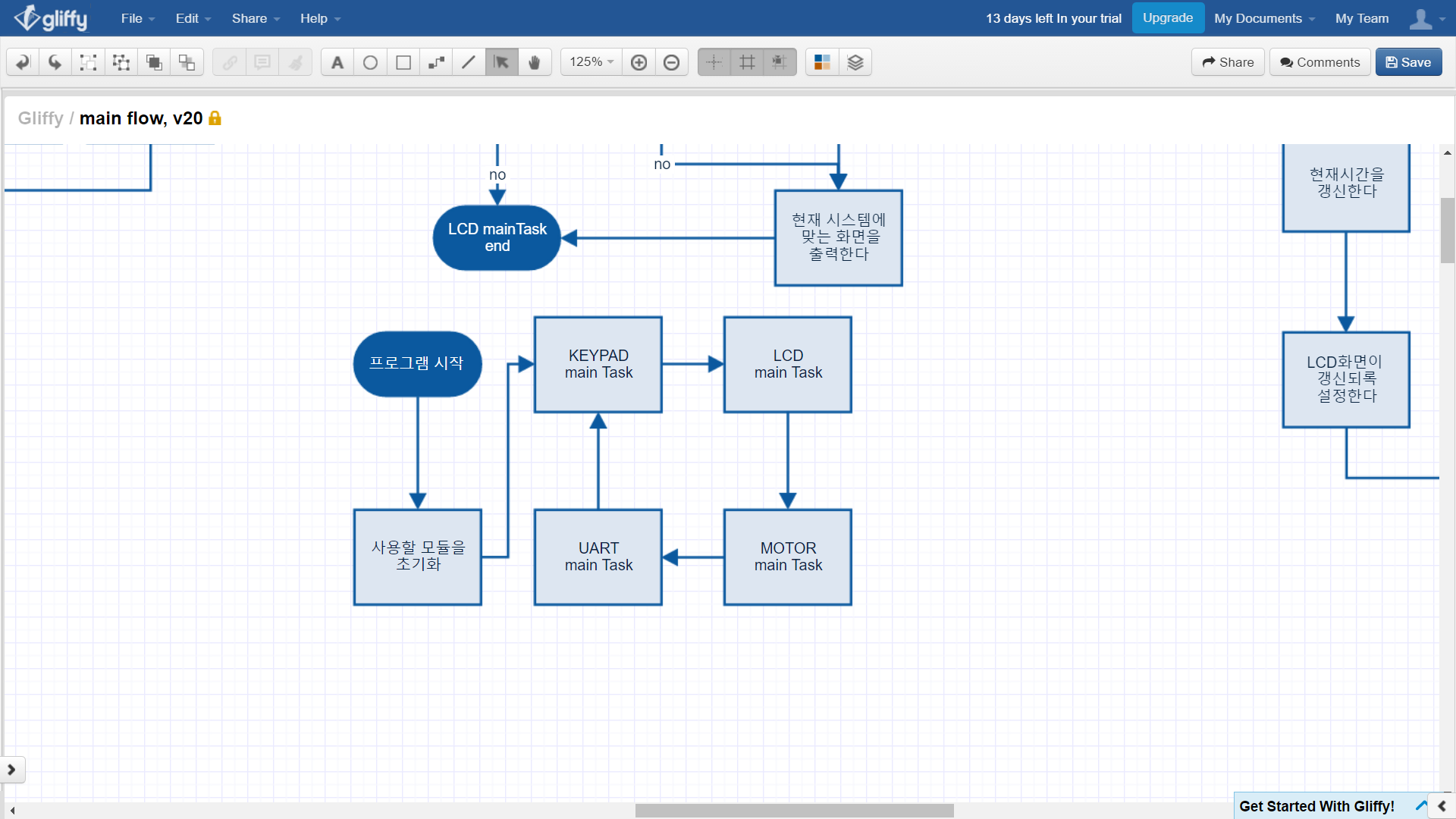
t.join()

except:

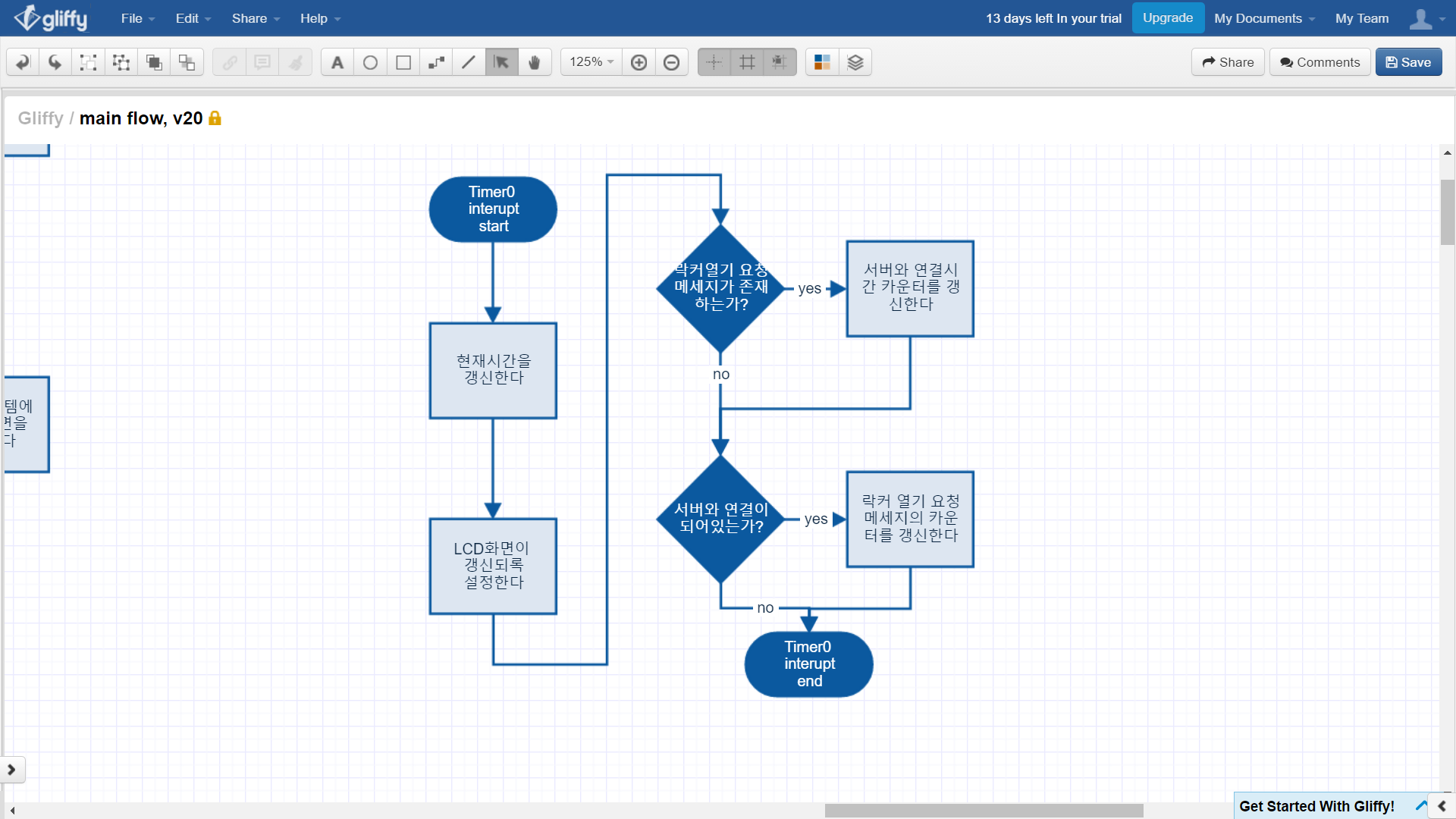
traceback.print\_exc(file = sys.stdout)

# Flow Chart

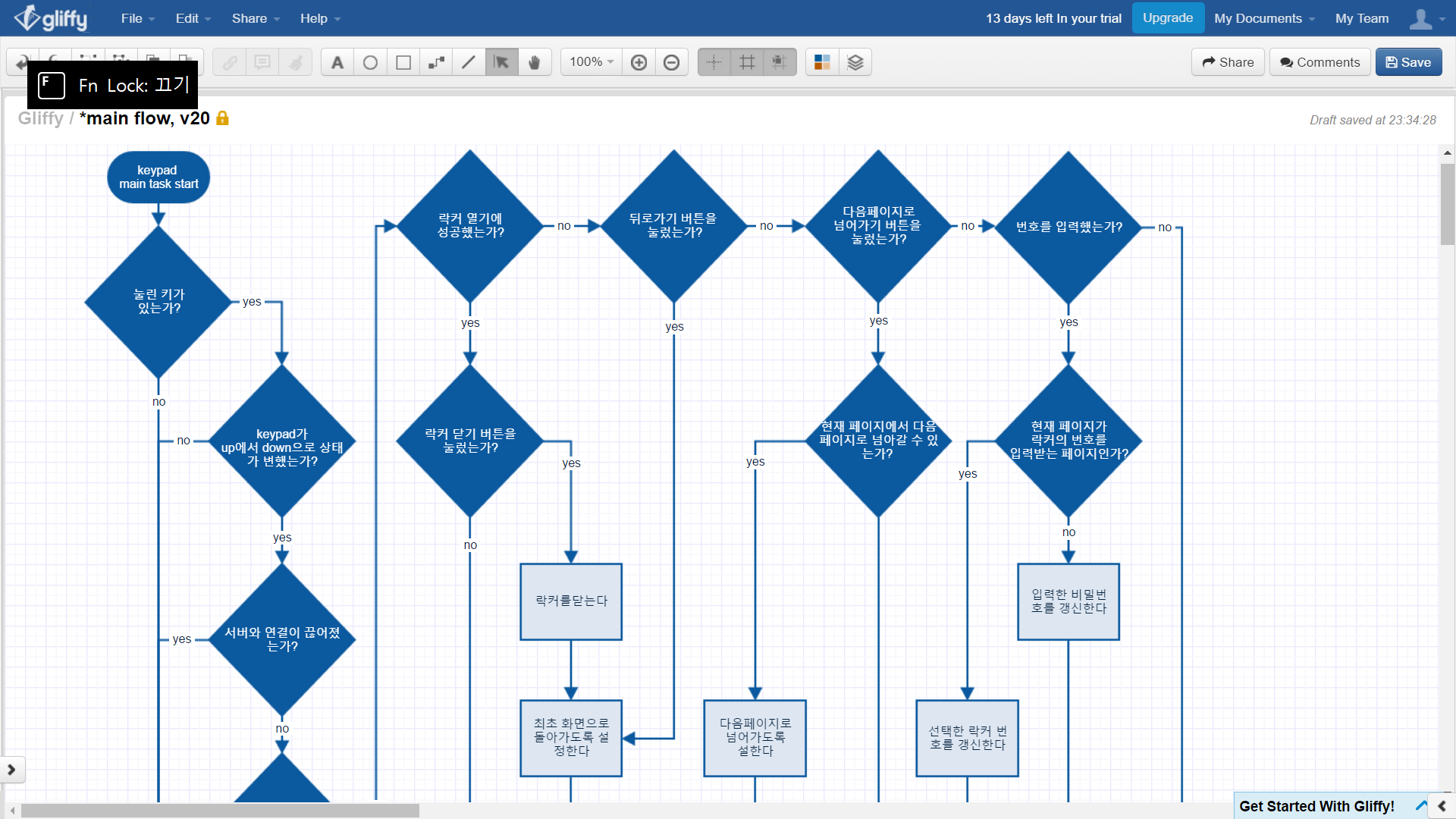
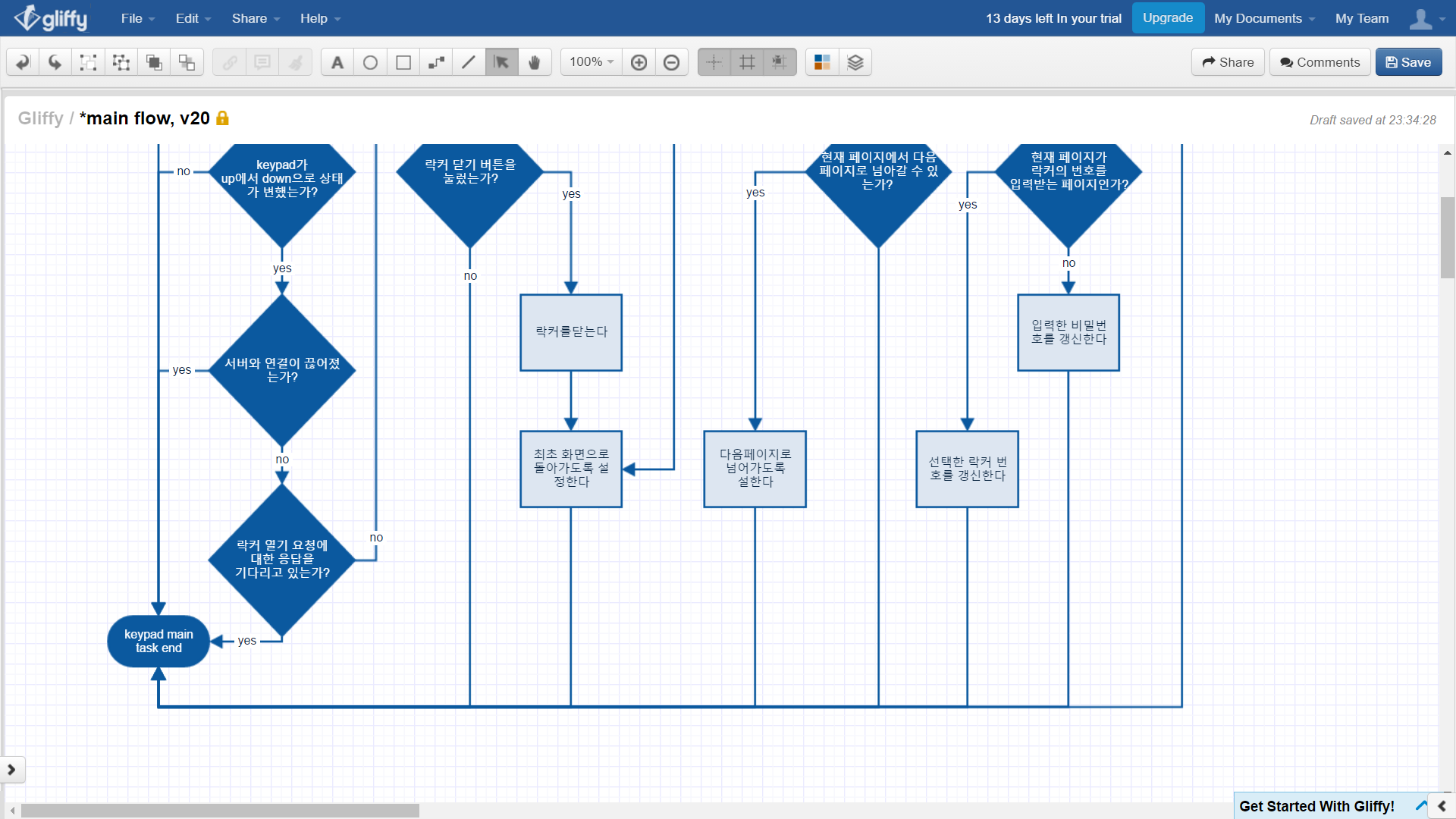
## ARM kit main Flow chart



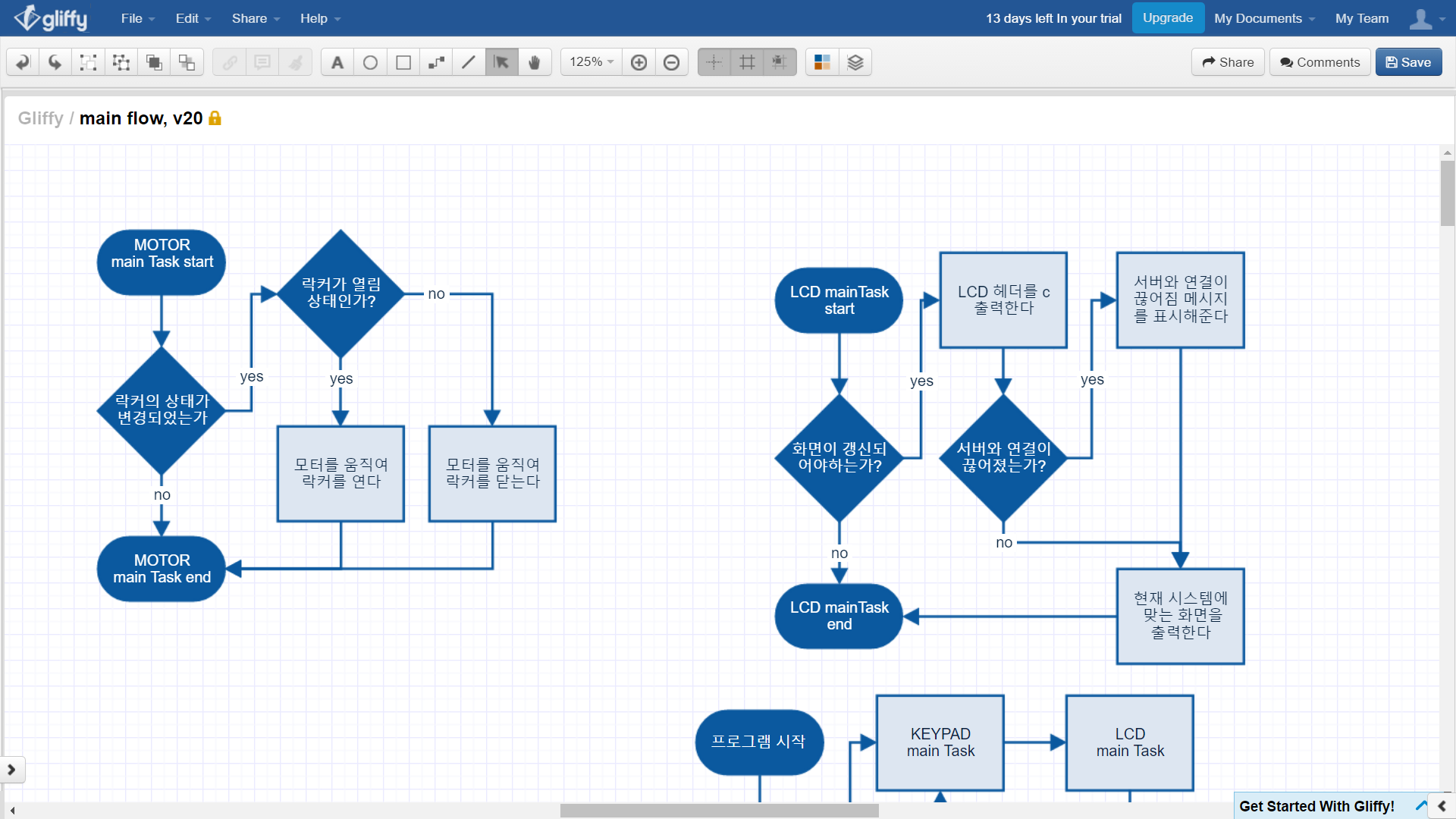
## Timer0 flow chart



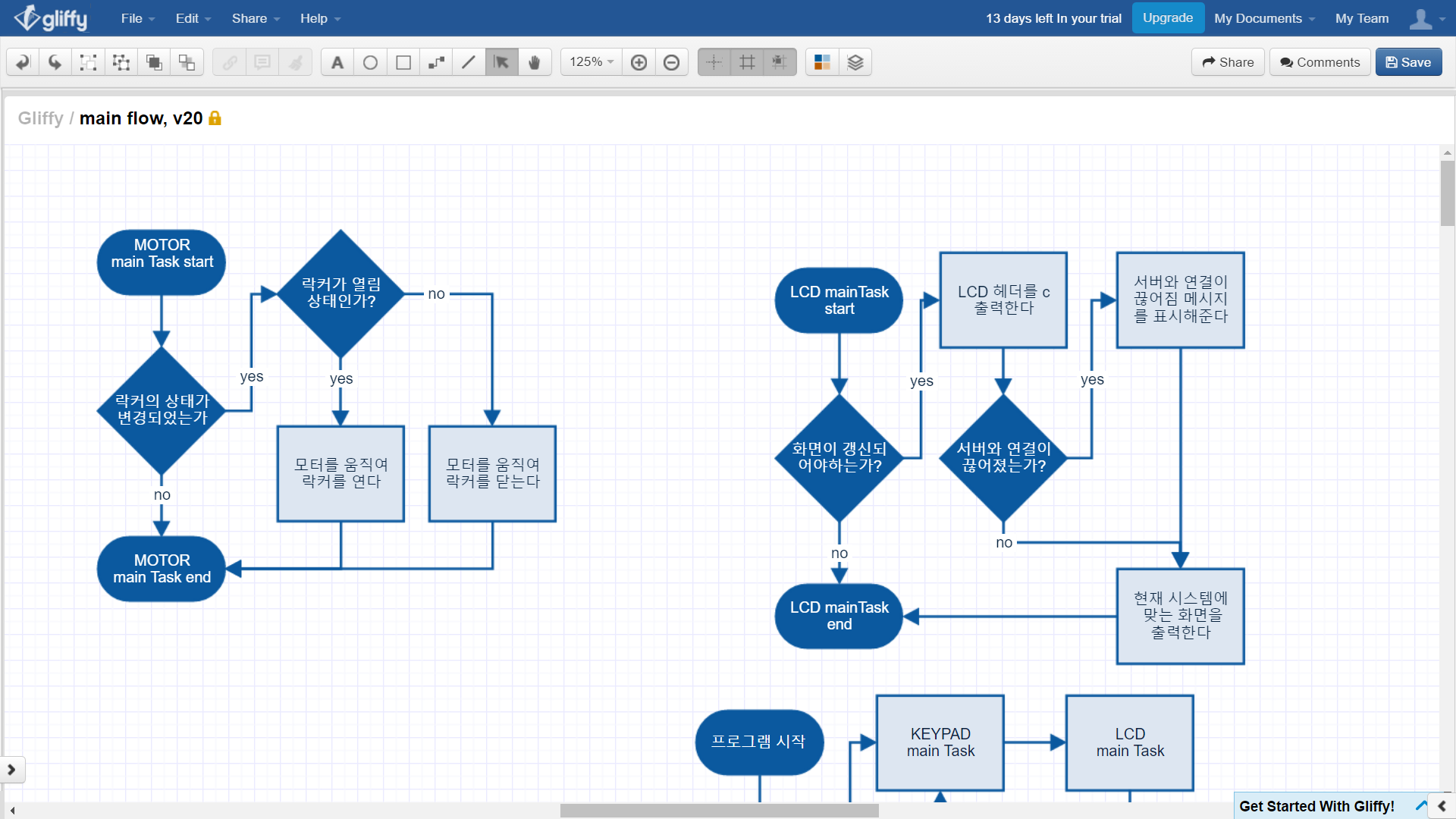
## KEYPAD main Task Flow chart



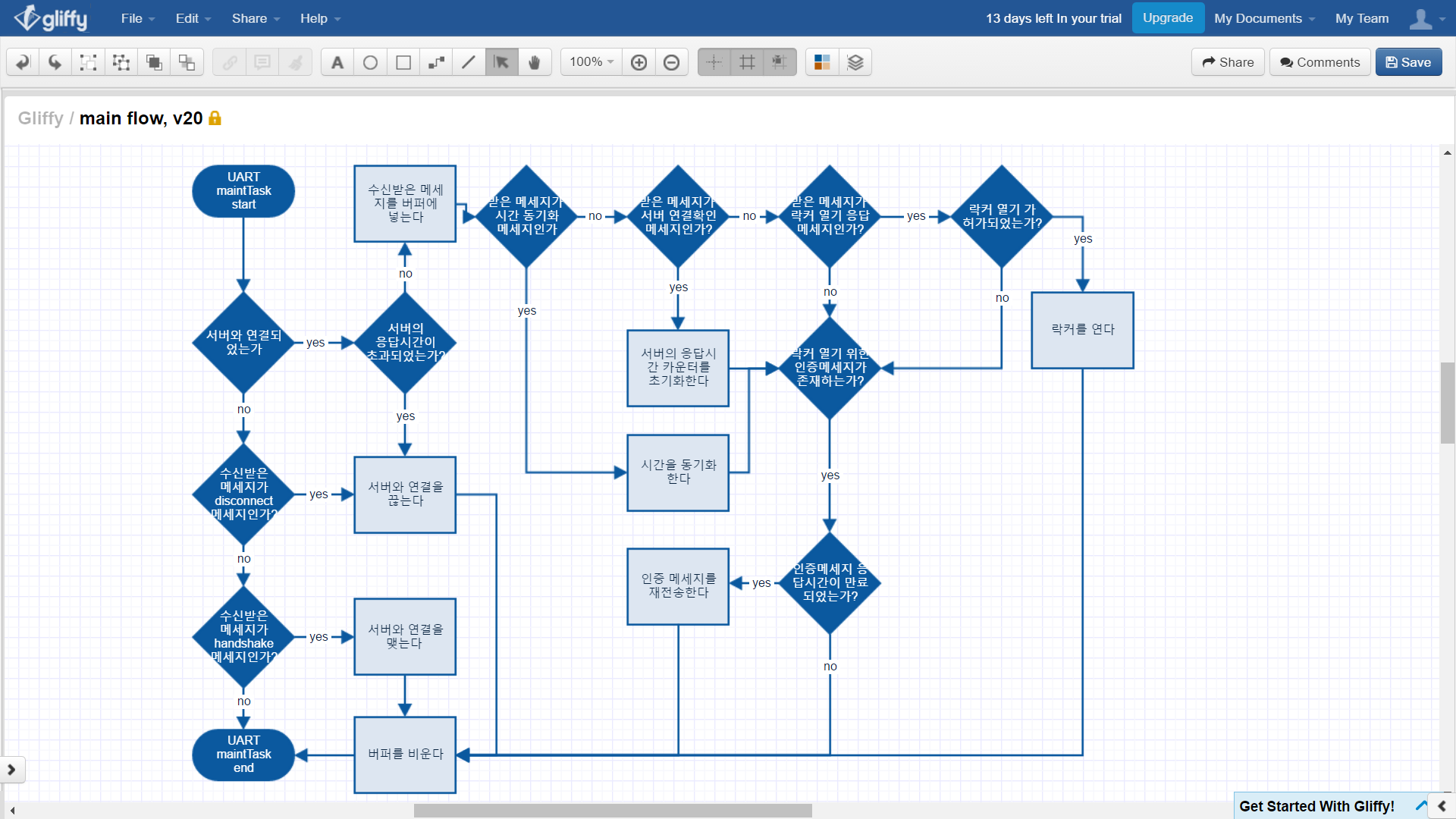
## Motor main Task flow chart



## LCD main Task flow chart



## UART main Task Flow chart



# 구현기능 설명

## ARM kit에서의 구현 기능

|  |  |
| --- | --- |
| KEYPAD | 다음에 나열한 입력을 받는다. 사물함 번호, 사물함 비밀번호, 입력 정보 지우기, 이전/다음 페이지 이동, OPEN/CLOSE |
| LCD | 날짜, 시간 표시  사물함 선택 화면, 비밀번호 입력 화면, 안내 메시지 출력 |
| Step Motor | 회전을 이용한 락커의 OPEN / CLOSE의 표현 |
| UART | Python ARM KIT server와의 기본적인 통신기능(handshaking, check connection, disconnect, reconnection)  서버와 ARM KIT의 시간 및 날짜 동기화  락커를 열기 위한 인증 요청 메시지 전송 및 수신 |

## flask web server & python arm kit server에서의 구현 기능

|  |  |
| --- | --- |
| Flask web server | Web 서버의 기본적인 로그인, 계정생성, 관리자 및 사용자 계정  락커 대여 및 반납  대여한 락커의 비밀번호 표시 |
| Python ARM kit server | ARM kit와 기본적인 통신기능(handshaking, check connection, disconnect, reconnection)  서버와 ARM kit의 시간 및 날짜 동기화  락커를 열기위한 인증 요청 메시지 처리  락커 열기 인증 성공 후 비밀번호 자동 갱신 |

# 조별 일지

|  |  |  |
| --- | --- | --- |
| 일시 | 참석자 | 내용 |
| 16.11.13(일) 16시 | 전원 | 키트 결정을 위한 토의 |
| 16.11.14(월) 19시 | 전원 | 키트 인수 후 고장여부 및 부품, 설명서 확인  주제 설정 토의  프로그램 μKEIL 설치 및 동작 확인 |
| 16.11.15(화) 19시 | 전원 | 키트와 개인 노트북 간의 연결 확인  예제 실습  예제 코드 분석 |
| 16.11.16(수) 19시 | 전원 | LED 동작 테스트  FND 동작 테스트  Step motor 및 Keypad의 물리적 인터페이스 제작  LCD 화면 구성 및 레이아웃 토의 |
| 16.11.22(화) 19시 | 전원 | LED, FND 동작 테스트 및 코드 구현, 수정  LCD 동작 확인 및 코드 구현, 수정  Keypad 동작 확인 |
| 16.11.23(수) 19시 | 전원 | Keypad 함수 구현  Main함수 정리 및 기능 구현  Step Motor 작동 확인 및 함수 구현 |
| 16.11.29(화) 19시 | 전원 | UART 통신  LCD 화면 출력 제어 (문자 출력 및 화면 갱신 점검)  시간 및 날짜 출력 제어 |
| 16.11.30(수) 19시 | 전원 | LCD 화면 출력 제어 및 정리 (화면 출력 범위 점검)  keypad 상세 기능 정리 |
| 16.12.04(일) 19시 | 전원 | 실 작동 테스트 및 오류사항 점검  Step motor 시계방향 회전구현을 위한 테스트 |
| 16.12.05(월) 17시 | 전원 | 최종 동작 확인 |

|  |  |  |
| --- | --- | --- |
| 일시 | 참석자 | 내용 |
| 16.11.17~21 | 김유준 | Step Motor 동작 구현  LCD 동작 구현 |
| 16.11.24~28 | 김유준 | UART 통신 구현  Python arm kit server 구현  Flask web server 구현  웹 페이지 구현 |
| 16.12.01~03 | 김유준 | ARM kit, Python ARM kit server, Flask web 서버 연동  동작테스트  Step motor 시계방향 회전구현을 위한 테스트 |

# 고장내역

없음

# 토의

## 구현하지 못한 기능

모터를 이용하여 락커의 OPEN/CLOSE를 구현하는 과정에서 스텝 모터를 반시계 방향으로 회전하는 것은 구현하였다. 그러나 원래 계획은 스텝모터를 시계방향과 반시계 방향 둘 다 움직여 락커의 OPEN/CLOSE를 구현하는 것에 있어 시계 방향 회전 기능의 신뢰성이 낮아 정확하게 구현하지 못하였다.

ARM kit의 스텝모터의 경우 4상 유니폴라의 스텝모터이다. 스텝모터 내부 2개의 전자석의 전류방향을 지정하여 스텝 시퀸스를 실행 것으로 회전을 하게 된다. 따라서 이론적으로 반시계 방향으로 회전 할 때의 스텝 시퀸스를 역순으로 하면 시계방향으로 회전 해야한다. 하지만 회전이 시계, 반시계 방향으로 번갈아 가면서 돌거나 시계 또는 반시계 방향만 회전하는 등 제어가 힘들었다. 시계 방향으로 회전하는 시퀸스 스텝을 찾기 위해 여러 차례의 실험을 통해서 스텝 모터의 특성을 알아내게 되었다.

* 온도가 낮은 경우 모터의 제어 원활 하지 못하다.
* 각 시퀸스 스텝 간에 모터를 제어를 위한 딜레이를 너무 짧게 한다면 모터의 내부의 회전자가 속도에 맞추지 못하여 제어가 원활 하지 못하다.
* 모터내부의 회전자와 회전축의 관성 때문에 정지했다가 움직이는 경우, 즉 속도 변화가 존재할 때 속도에 맞추어 모터제어를 위한 딜레이를 다르게 해야 제어가 원활하다.
* 모터의 제어를 위한 딜레이가 적절하지 못하다면 외부에서 회전축에 현재 회전 방향의 반대 방향으로 힘을 가하면 회전 방향이 변하기도 한다.
* 모터 외부에서는 내부의 회전자의 상태를 관찰할 수 없으므로 회전축이 동일하더라도 결과가 달라질 수 있다.

이와 같은 특성들로 인하여 소프트웨어 적으로 방법으로 모터회전을 제어 하는 것은 어려웠다. 하드웨어 적으로 스텝 모터의 6개의 입력선을 다르게 구성하거나 키트에 직접 회로를 추가하여 하는 방법도 고려하였지만 적절한 장비를 구할 수가 없고 파손의 우려가 있어 시도할 수 없었다. 그래서 시연 발표를 전날에도 모터의 시계방향 회전 작동의 신뢰도를 높이기 위해 노력했으나 최종적으로는 반시계방향 회전만으로 구현하는 방식을 선택하였다.

## 계획에 비해 새롭게 추가한 기능

원래 계획은 Python ARM kit server까지만 구현하는 것이지만 사용자 접근성을 위하여 GUI 인터페이스가 필요했다. 그래서 Python 기반으로 만들어진 Flask 라는 Mircroframework을 사용하였다.

## 계획에 비해 제외된 기능

이 프로젝트는 초기 전자 도어락 또는 전자 출석기를 구현하는 것이었다. 이 당시 FND(7segment)는 시간 부분을 표시하거나 기타 메시지를 띄울 용도였으나 LCD의 기능과 겹치고 LCD의 느린 IO 속도를 고려해야 했다. FND는 하나의 GPIO를 사용하여 제어하게 되는데 이때 8개의 FND중 한번에 하나의 FND만 제어가 가능하다. 따라서 매우 짧은 시간동안 순서대로 하나씩 FND를 점등하는 것을 반복하는 것으로 잔상효과 발생시켜 동시에 켜진 것 같은 착시 현상을 일으켜서 구현하는 방식을 사용해야했다. 또한 무엇보다 중요한 것은 이 프로젝트에서 대부분의 출력기능은 LCD가 대신할 수 있다는 점 때문에 제외시켰다.

## final discussion

이 팀프로젝트에서는 ARM cortex processor를 이용하여 디지털 사물함식 냉장고 관리시스템를 마이크로 프로세스를 이용한 프로그램 설계와 하드웨어를 직접 제어 및 UART를 이용한 시리얼 통신을 실습 할 수 있었다.

ARM kit 프로그램 설계시에는 Timer0, LCD, Keypad 파트에서는 ARM kit에서 제공하는 질 높은 예제 프로그램과 라이브러리를 사용하여 쉽게 설계할 수 있었다. UART의 경우 라이브러리 중 바이트 스트림을 수신 하는 함수가 작동을 하지않아서 직접 구현하였고 Stepmotor의 경우 시계방향 회전구현에 골머리를 썩혔다. 무엇보다도 팀 프로젝트이므로 다른 팀원들도 쉽게 코딩할 수 있도록 코드의 구조와 함수 및 변수이름을 만드는데 심혈을 기울였고 지속적으로 코드리뷰를 통하여 팀원들과 코드의 이해도를 높이는데 노력을 하였다.

Python ARM kit server를 구현하는데 있어서 이미 시리얼 통신 라이브러리가 존재하기 때문에 polling 방식으로 전송하는 부분을 제외하면 편리하게 구현할 수 있었다. Flask web server부분의 경우 원래 계획에 포함되어 있지않았으나 사용자의 접근성을 고려하여 추가하였고 Python기반이 때문에 Python ARM kit server와의 연동 또한 쉽게 구현하였다.

이 프로젝트를 하면서 키트자체의 잔고장이나 결함으로 인한 시간낭비가 없었기 때문에 구현목표를 거의 모드 구현하는 것이 가능하였다. 하지만 무엇보다도 팀원들이 열심히 참여했기 때문에 프로젝트를 마무리 할 수 있었다. 팀프로젝트에 참여한 모든 팀원에게 감사를 표한다.