# 마이크로 프로세서

4조 First Spring
201613989 김진산
201610349 최성우
201822483 정 우

전북대학교
CHONBUK NATIONAL UNIVERSITY

<제목 차례>



<그림 차례>

# 1. 프로젝트 개요

## 1.1 프로젝트 주제

프로젝트의 주제는 간단한 슈팅 게임입니다. 게임의 구성은 4가지로 이루어져 있습니다. 플레이어가 조종하는 비행기인 Fighter, 파괴해야하는 장애물인 Meteor, 비행기가 발사하는 Missile, 장애물을 파괴한 수인 Point로 구성이 됩니다.

게임 규칙은 LCD의 왼쪽에 위치한 Fighter를 조종하여 오른쪽 끝에서 날아오는 Meteor를 파괴하는 것입니다. Fighter는 상, 하로만 이동이 가능하며 Missile을 발사할 수 있습니다. Meteor를 파괴한다면 1의 점수를 얻을 수 있습니다. Meteor를 파괴하면 다음 Meteor는 좀 더 빠른 속도로 이동하게 됩니다. 게임이 종료되는 규칙은 2가지입니다. Fighter가 Meteor와 충돌하여 부서지는 경우, Meteor를 3번 파괴하지 못하는 경우입니다. 게임이 종료되면 점수와 함께 종료 화면이 등장하게 됩니다.

게임은 LCD와 Joystick을 사용하여 진행됩니다. LCD는 게임 화면으로 사용이 됩니다. joystick을 통해서 플레이어는 Fighter를 조종할 수 있습니다. joystick을 상, 하로 조작하여 플레이어는 Fighjter를 상, 하로 이동시킬수 있습니다. joystick을 우측으로 조작하면 Missile을 발사할 수 있습니다.

아래 그림은 간단하게 표현한 게임의 목업입니다.

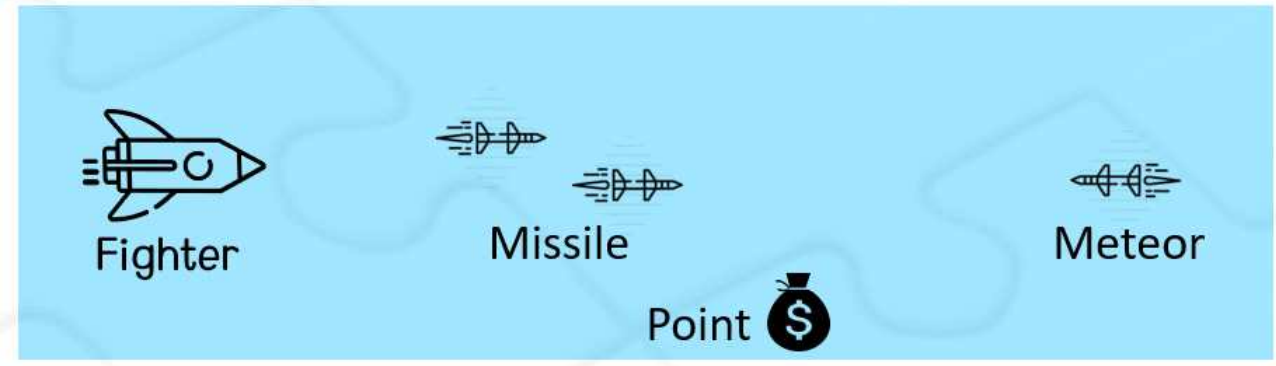


그림 1 게임 목업

2. 실행 화면

게임을 구성하는 화면은 총 3개입니다. 게임 화면, Fighter 충돌시 게임 오버, Meteor 3개 놓칠 경우 게임 오버 화면입니다. 게임 화면은 Fighter (>), Missile(-), Meteor(O)와 하단에 게임 점수, 목숨으로 구성되어 있습니다. Meteor 파괴 및 놓칠 경우 점수, 목숨의 값이 변합니다.

게임 종료 조건에 따라 게임 종료 화면은 각각 종료 조건을 LCD에 표시하는 동시에 점수도 표시합니다.

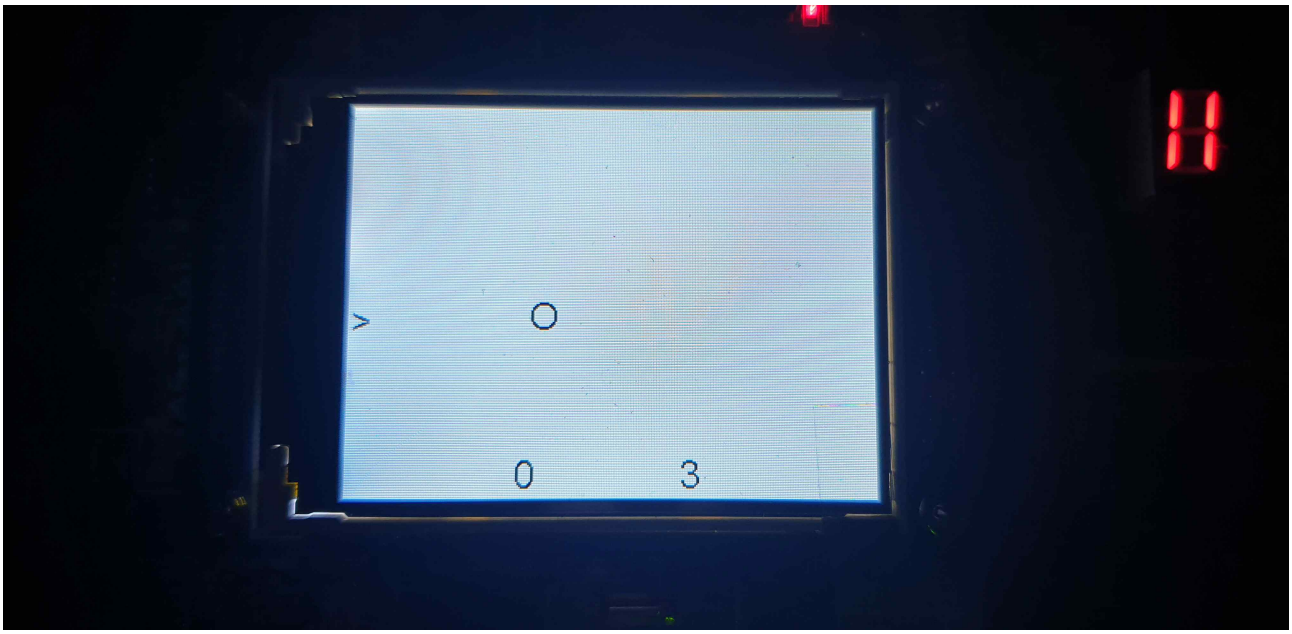


그림 2 기본 게임 화면

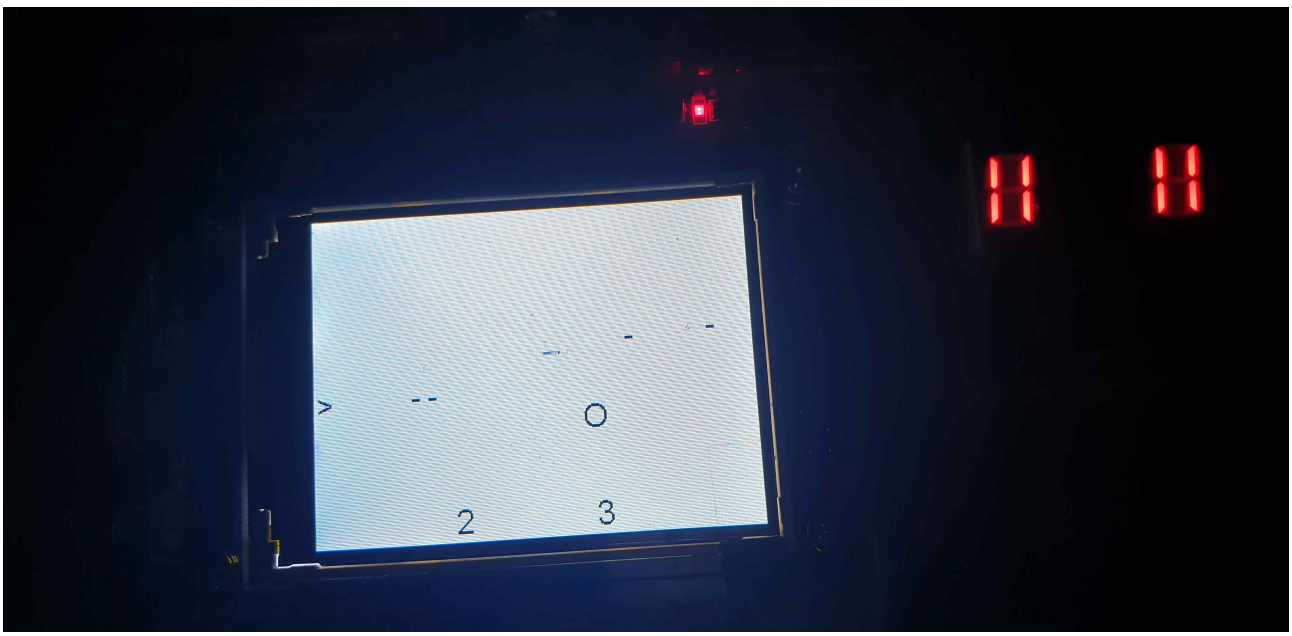


그림 3 Missile 발사 화면

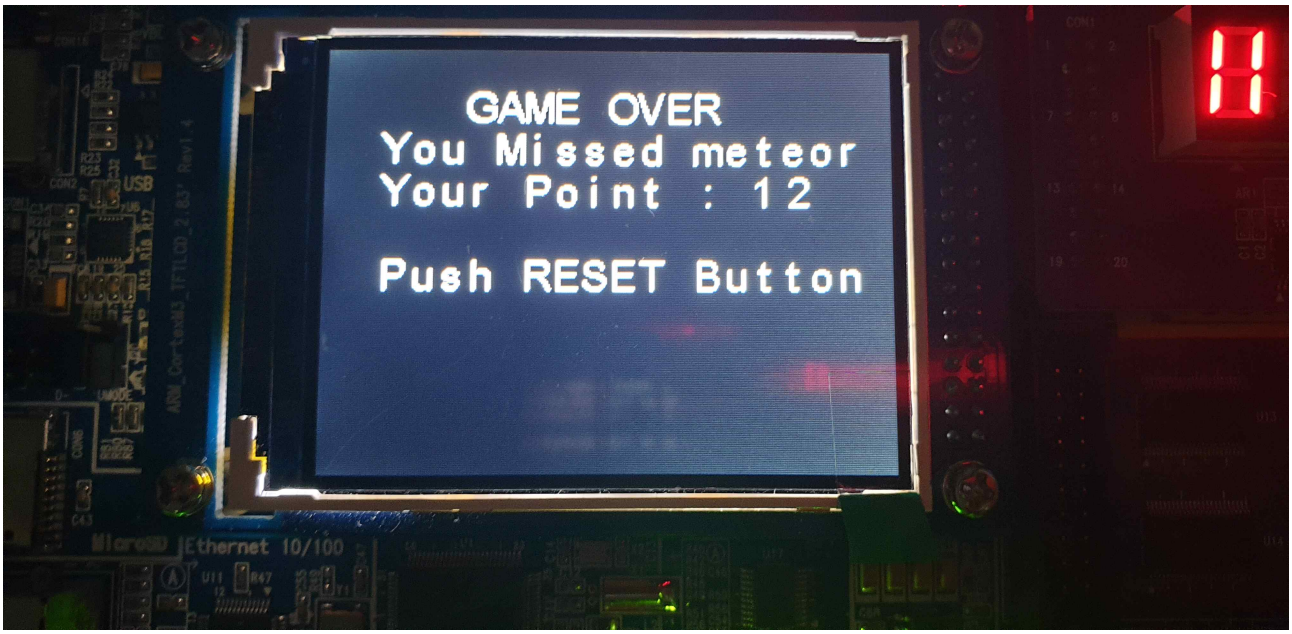


그림 4 Meteor 놓쳤을 때 gameOver

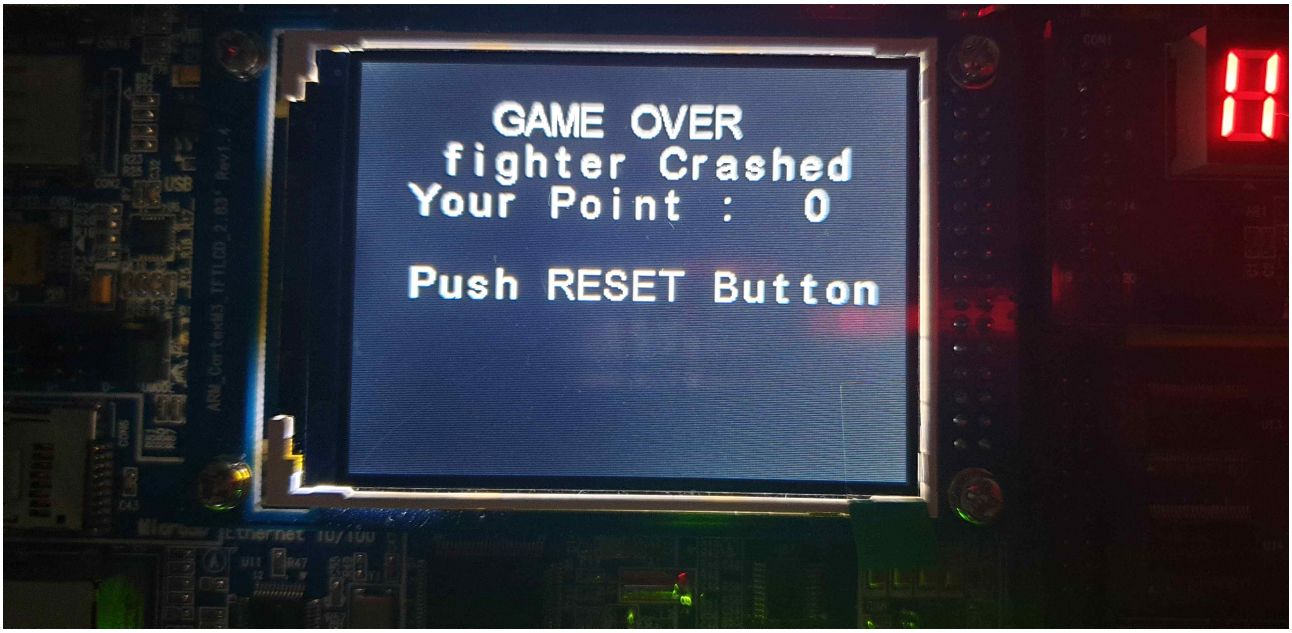


그림 5 Fighter 충돌시 gameOver

## 3. 문제 발생

### 3.1 Joystick의 연속된 입력

조이스틱을 사용할 때 1번이 아닌 연속적으로 계속 눌리는 경우가 발생하여 Fighter의 조작이 거의 불가능 했습니다. 상, 하로 이동하면 맨위, 맨 아래로 바로 이동하게 되었습니다.

해결하려고 조이스틱을 조정하는 함수에 key값을 초기화 해서 연속된 입력을 방지하려 했지만 변화가 없었습니다. 함수의 실행 속도가 빨라서 발생한 현상이라 생각이 됩니다.

이후 delay함수로 joystick입력후 잠깐의 시간을 만들어서 다중 입력을 방지하려 했지만 1초정도는 게임 진행속도에 영향이 많이 있는 것 같아 for문을 사용해서 속도를 감소시켰습니다. for문의 반복 횟수를 계속 변화시켜서 실행해 봄으로써 눈으로 가장 적절하다 느낀 수로 정했습니다. Missile이 아직 한번에 여러개 나가는 현상이 있지만 이 이상 쉬는 시간을 주웠을 때는 게임 속도에 영향이 있어서 너무 많이 쉬도록은 하지 않았습니다.

### 3.2 Missile 개수에 따른 게임 랙 발생

초기 Missile 설정의 100개로 잡았습니다. LCD화면에 표시된 Missile을 1픽셀씩 하나하나 이동을 해서 무리가 있었는지 게임에 랙이 걸렸습니다. 그래서 Missile의 개수를 50개로 줄였다가 같은 이유로 20개로 제한을 했습니다.

아마도 LCD에 문자를 표시하는데 부하가 조금 있는 거라 판단이 됩니다.

### 3.3 충돌 범위

단순하게 x, y좌표로 충돌을 판단하기 어려움이 있었습니다. miisile의 크기, fighter의 크기, meteor의 크기를 알 수 있는 방법을 찾지 못해서 대략적으로 16~20으로 계산해서 조건을 만들고 실행시켜봐서 가장 적절한 조건을 경험으로 찾았습니다.

### 3.4 rand 함수

c에서 rand()를 사용할때는 항상 srand()를 이용해서 rand()의 결과값이 바뀌도록 해야 했는데, srand()가 오류가 발생했습니다. 그러나 rand()가 잘 동작하는 것을 확인했습니다. 이유는 아직 모르겠습니다.

### 3.5 점수 표기 방법

int형 변수를 직접적으로 LCD에 표기할 수 없어서 형 변환을 통해서 char로 변환하여 자리 수마다 하나씩 표기하는 방법을 사용하였습니다. 999까지는 글자 겹침이 없이 잘 표기가 될꺼라 판단 됩니다.

## 4. 알게 된점

### 4.1 LCD의 한 문자의 픽셀 크기

darw_char의 함수의 파라미터로 보아 x는 16, y는 24픽셀을 하나의 문자의 크기로 설정하고 있음을 알 수 있습니다. 또한 y의 경우 LINE의 정의를 보아 24픽셀이라 유추할 수 있습니다.

### 4.2 GLCD_displayChar에서 c-=32의 이유

c -= 32이유: ASCII_TABLE의 시작이 space 즉 아스키 코드 32부터 다루기 때문에 필요합니다. c *24의 경우도 ASCI_TABLE의 배열이 24개로 구성되어 있어서 필요하다고 유추할 수 있습니다.

아래 코드 분석을 하면서 많이 찾아보면서 많이 알게된 것 같습니다.

## 5. LCD 코드 분석

### 5.1 GLCD_setTextColor

```
void GLCD_setTextColor(unsigned short color){
 TextColor = color;
}
```

```
static volatile unsigned short TextColor = Black, BackColor = White;
```

color를 받아서 TextColor를 변환해주는 것으로 LCD의 글자 색을 변경해준다.
volatile로 선언된 변수는 항상 메모리에 접근합니다.

### 5.2 GLCD_setBackColor

```
void GLCD_setBackColor(unsigned short color){
 BackColor = color;
}
```

```
static volatile unsigned short TextColor = Black, BackColor = White;
```

setTextColor와 마찬가지로 color를 받아서 BackColor를 변경시켜주는 것으로 LCD의 배경 색을 변경합니다.

### 5.3 GLCD_clear

```
void GLCD_clear (unsigned short color){
 unsigned int i;
 GLCD_SetX(0, WIDTH);
 GLCD_SetY(0, HEIGHT);
 wr_cmd(0x2C); // RAM Write
 for(i = 0; i < (WIDTH*HEIGHT); i++) wr_pixel(color);
}
```

color를 받아 LCD의 모든 pixel의 색을 변환 시키는 함수

### 5.4 GLCD_SetX

```
static void GLCD_SetX(unsigned int x, unsigned int width){
 wr_cmd(0x2A);
 wr_dat((x>>8) & 0xFF); //8~15
 wr_dat(x & 0xFF); //0~7
 wr_dat(((x+width-1)>>8) & 0xFF); //8~15
 wr_dat((x+width-1) & 0xFF); //0~7
}
```

x좌표와 넓이를 설정하는 함수로 판단됨

### 5.5 wr_cmd

```
/*********************************************************************************
```

```
* Write command to LCD controller                                          *
*    Parameter:      c:        command to be written                       *
*    Return:                                                               *
*****************************************************************************/
static __inline void wr_cmd (unsigned char c){
 LPC_GPIO0->FIOPIN &= ~(1 << 6);
 ssp_send(c | SSP_INDEX);
 LPC_GPIO0->FIOPIN |=  (1 << 6);
}
```

| | | | | |
|---|---|---|---|---|
| FIOPIN | Fast Port Pin value register using FIOMASK. The current state of digital port pins can be read from this register, regardless of pin direction or alternate function selection (as long as pins are not configured as an input to ADC). The value read is masked by ANDing with inverted FIOMASK. Writing to this register places corresponding values in all bits enabled by zeros in FIOMASK.<br>**Important:** if an FIOPIN register is read, its bit(s) masked with 1 in the FIOMASK register will be read as 0 regardless of the physical pin state. | R/W | 0x0 | FIO0PIN - 0x2009 C014<br>FIO1PIN - 0x2009 C034<br>FIO2PIN - 0x2009 C054<br>FIO3PIN - 0x2009 C074<br>FIO4PIN - 0x2009 C094 |

그림 6 FIOPIN

FIOPIN은 데이터 읽기, 쓰기에 사용된다.

LCD 컨트롤러에 명령을 하는 함수

```
LPC_GPIO0->FIODIR   |=  (1 <<  6);    /* Pin P0.6 is GPIO output (SSEL1)    */
LPC_GPIO0->FIOSET    =  (1 <<  6);    /* Set P0.6 high                      */
```

그림 7 GPIO0 설정

| | | | | |
|---|---|---|---|---|
| FIODIR | Fast GPIO Port Direction control register. This register individually controls the direction of each port pin. | R/W | 0x0 | FIO0DIR - 0x2009 C000<br>FIO1DIR - 0x2009 C020<br>FIO2DIR - 0x2009 C040<br>FIO3DIR - 0x2009 C060<br>FIO4DIR - 0x2009 C080 |

그림 8 FIODIR

FIODIR: 레지스터의 방향 제어하는 핀

| | | | | |
|---|---|---|---|---|
| FIOSET | Fast Port Output Set register using FIOMASK. This register controls the state of output pins. Writing 1s produces highs at the corresponding port pins. Writing 0s has no effect. Reading this register returns the current contents of the port output register. Only bits enabled by 0 in FIOMASK can be altered. | R/W | 0x0 | FIO0SET - 0x2009 C018<br>FIO1SET - 0x2009 C038<br>FIO2SET - 0x2009 C058<br>FIO3SET - 0x2009 C078<br>FIO4SET - 0x2009 C098 |

그림 9 FIOSET

FIOSET : 출력핀 상태 제어

LPC_GPIO0->FIOPIN &= ~(1 << 6);

- 6번 핀을 0으로 변경한다.
- FIOMASK에서 0으로 활성화된 비트만 변경할 수 있음

ssp_send(c | SSP_INDEX);

- c | SSP_INDEX : or비트 연산을 하지만 ssp_index가 0x00이므로 c가 그대로 파라미터로 입력됨
- SSP_INDEX : 명령어 레지스터임을 표시한다.(8번째 비트가 0, wr_dat과 비교시 RS 신호선이 0으로 판단됨)
- SSP : 동기식 직렬 포트 (SSP)는 그 컨트롤러 지원 직렬 주변기기 인터페이스 (SPI), 4 선 동기식 시리얼 인터페이스 (SSI)을 마이크로 와이어 직렬 버스 . SSP는 마스터-슬레이브 패러다임을 사용하여

연결된 버스를 통해 통신 합니다.
LPC_GPIO0->FIOPIN |= (1 << 6);
- 6번 핀을 1로 변경한다.

## 5.6 ssp_send

```
/*****************************************************************************
* Send 1 byte over serial communication                                      *
*    Parameter:      byte:    byte to be sent                                *
*    Return:                                                                 *
*****************************************************************************/
static __inline unsigned char ssp_send (unsigned int byte){
 LPC_SSP1->DR = byte & 0x1FF;
 while (LPC_SSP1->SR & (1 << 4));       /* Wait for transfer to finish       */
 return (LPC_SSP1->DR) & 0xFF;          /* Return received value             */
}
```

| | | | | |
|---|---|---|---|---|
| DR | Data Register. Writes fill the transmit FIFO, and reads empty the receive FIFO. | R/W | 0 | SSP0DR - 0x4008 8008<br>SSP1DR - 0x4003 0008 |

UM10360_0 © NXP B.V. 2009. All rights reserved.

그림 10 SSP1 - DR

| | | | |
|---|---|---|---|
| SR | Status Register | RO | SSP0SR - 0x4008 800C<br>SSP1SR - 0x4003 000C |

그림 11 SSP1 - SR5

시리얼 통신을 통해 1바이트 전송

Master와 Slave mode에서의 전이중 통신 송수신 절차(교제 P361) - SSP는 마스터-슬레이브 패러다임을 사용하여 연결된 버스를 통해 통신

LPC_SSP1->DR = byte & 0x1FF;
 TXE를 1로 만들고 (8번째 비트로 추정) 8개의 bit를 전송
while (LPC_SSP1->SR & (1 << 4));
 BSY가 0이 될 때까지 대기한다 (BSY는 4번째 비트로 추정)
return (LPC_SSP1->DR) & 0xFF;
 TXE를 0으로 만든다.
 DR의 8개의 비트를 반환함 (1바이트).

## 5.7 wr_dat

```
/*****************************************************************************
* Write data to LCD controller                                               *
*    Parameter:      c:       data to be written                             *
*    Return:                                                                 *
*****************************************************************************/
static __inline void wr_dat (unsigned short c)
{
```

```
        LPC_GPIO0->FIOPIN &= ~(1 << 6);
        ssp_send(c | SSP_DATA);     /* Write : RS = 1, RW = 0        */
        LPC_GPIO0->FIOPIN |=  (1 << 6);
}
```

LCD 컨트롤러에 데이터를 쓰는 함수

```
ssp_send(c | SSP_DATA);
 SSP_DATA : 0x100
 or연산으로 8번째 비트는 항상 1이 된다. -> 데이터 레지스터임을 알림(RS신호선) ,TXE Flag로 판단됨.
```

RS신호선은 내부의 레지스터를 엑세스할 때, 명령 레지스터인지 (RS=0), 데이터 레지스터인지를 (RS=1) 구별하는데 사용됩니다.
RW는 읽는 동작에서는 이 신호가 L(0)이고 쓰는 동작에서는 H(1)가 됩니다.
Tx Buffer에 쓰기를 시도하기 전에 TXE Flag가 1인지 확인해야 한다. (교제 P360)

## 5.8 GLCD_SETY

```
static void GLCD_SetY(unsigned int y, unsigned int height){
 wr_cmd(0x2B);
 wr_dat((y>>8) & 0xFF); //8~15
 wr_dat(y & 0xFF); //0~7
 wr_dat(((y+height-1)>>8) & 0xFF); //8~15
 wr_dat((y+height-1) & 0xFF); //0~7
}
```

y좌표와 높이를 설정하는 함수로 판단됨

## 5.9 wr_pixel

```
/*********************************************************************************
* Write Pixel data to LCD controller                                             *
*    Parameter:      c:        Pixel data to be written                          *
*    Return:                                                                     *
*********************************************************************************/
static __inline void wr_pixel (unsigned short c){
 LPC_GPIO0->FIOPIN &= ~(1 << 6);
 ssp_send((c >> 8) | SSP_DATA);          /* Write D8..D15                */
 ssp_send((c & 0xFF) | SSP_DATA);         /* Write D0..D7                 */
 LPC_GPIO0->FIOPIN |=  (1 << 6);
}
```

LCD 컨트롤러에 픽셀 데이터 쓰는 함수로 판단됨

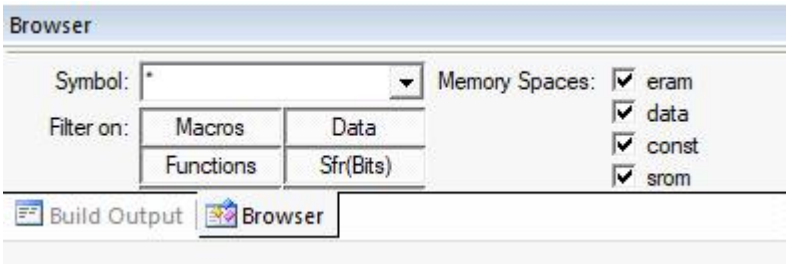


그림 12 wr_pixel go to definition 실행 화면

## 5.10 GLCD_displayChar

```
void GLCD_displayChar(unsigned int x, unsigned int y, unsigned char  c){
 c -= 32;
 GLCD_drawChar(x, y, &ASCII_Table[c * 24]);
}
```

c -= 32이유: ASCII_TABLE의 시작이 space 즉 아스키 코드 32부터 다루기 때문에 필요

c *24의 경우도 ASCI_TABLE의 배열이 24개로 구성되어 있어서 필요

```
24 uint16_t ASCII_Table[] = {
25   /* Space ' ' */
26   0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
27   0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
28   0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
29   /* '!' */
30   0x0000  0x0180  0x0180  0x0180  0x0180  0x0180  0x0180  0x0180
```

그림 13 ASCII_TABLE

## 5.11 GLCD_drawChar

```
/*********************************************************************************
* Draw character on given position                                               *
*    Parameter:      x :         horizontal position                             *
*                    y :         vertical position                               *
*                    c*:         pointer to color value                          *
*    Return:                                                                     *
*********************************************************************************/
void GLCD_drawChar(unsigned int x, unsigned int y, unsigned short *c){
 unsigned int index = 0;
 int  i = 0;
 GLCD_SetX(x, 16);
 GLCD_SetY(y, 24);
 wr_cmd(0x2C);
 for(index = 0; index < 24; index++){
  for(i = 0; i <16; i++){
   if((c[index] & (1 << i)) == 0x00){
    wr_pixel(BackColor);
```

```
    }
    else{
      wr_pixel(TextColor);
    }
   }
  }
}
```

LCD 화면에 x, y 좌표에 입력하는 함수
넓이는 16, 높이는 24로 설정했음을 알수 있다.

## 5.12 GLCD_displayStringLn

```
void GLCD_displayStringLn(unsigned int y, char *s){
 unsigned int i = 0;
 unsigned int refcolumn = 0;
 while ((*s != 0) & (i < 20)){    /* write the string character by character on lCD */
  GLCD_displayChar(refcolumn, y, *s);          /* Display one character on LCD */
  refcolumn += 16;                              /* Decrement the column position by 16 */
  s++;                                         /* Point on the next character */
  i++;                                         /* Increment the character counter */
 }
}
```

문자열을 한줄에 입력하는 함수
문자열을 받아서 문자 하나하나를 refcolumn이라는 넓이 값을 이용하여 표시한다.

## 6. 코드

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <RTL.h>
#include <LPC17xx.H>                    /* LPC17xx definitions                */
#include "GLCD.h"
#include "LED.h"

#define KBD_SELECT       0x01
#define KBD_LEFT         0x08
#define KBD_UP           0x10
#define KBD_RIGHT        0x20
#define KBD_DOWN         0x40
#define KBD_MASK         0x79

// 조이스틱 움직임 설정
#define right 1
#define left 2
#define up 3
#define down 4
//미사일 개수 설정
#define missile_max 20

int key; //조이스틱 이동 방향
int fighter_posX; //비행기 x좌표
int fighter_posY; //비행기 y좌표
int meteor_posX; //장애물 x좌표
int meteor_posY; //장애물 y좌표
int meteor_speed; //장애물 이동 속도

int inGame; //게임 진행 표시
int isShoot; //장애물 파괴 확인
int isPassed; //장애물 놓친 개수
int isMissile[50] = {0}; //미사일 발사 중 확인
int missile_posX[100]; //미사일 x좌표
int missile_posY[100]; //미사일 y좌표
int missile_num; //미사일 개수

int point; //점수

char fighter_shape; //비행기 모양
char meteor_shape; //장애물 모양
```

```
char missile_shape; //미사일 모양

//장애물 생성
void createmeteor(){
 meteor_posX = 300; //장애물 시작 위치 x좌표 오른쪽 끝
 meteor_posY = (rand()%Line7) + 24;    //장애물 시작 y좌표, 랜덤으로 생성 (+24는 크기 고려)
 GLCD_displayChar(meteor_posX, meteor_posY, meteor_shape);  //LCD에 표시
}
//미사일 이동
void missileMove(){
 int i =0;
 while(i<missile_max){   //모든 미사일 탐색
  if(isMissile[i]==1){ //현재 날라가는 미사일
   if(meteor_posY-12  <=  missile_posY[i]  &&  meteor_posY+12  >=  missile_posY[i]  &&
meteor_posX <=missile_posX[i]){ //미사일과 장애물 충돌 조건
    int tmpp;
    int x = 100, y = Line9;
    isMissile[i] = 0; //미사일 제거
    GLCD_displayChar(meteor_posX, meteor_posY, ' ');//장애물 LCD에서 제거
    GLCD_displayChar(missile_posX[i], missile_posY[i], ' '); //미사일 LCD에서 제거
    createmeteor(); //장애물 생성
    meteor_speed*=1.1; //장애물 이동 속도 증가
    point++; //포인트 증가
    //LCD화면에 점수 표기
    tmpp = point;
    while(tmpp>0){ //int형 표시를 위해 형변환 작업
     char tmp = (char)(tmpp%10+'0') //1의 자리 형변환
     GLCD_displayChar(x, y, tmp);
     x-=20; //LCD표시 위치 변경 (1의 자리부터 표기하기 때문에 -20)
     tmpp = tmpp/10;
    }
    break;
   }
   else{ //미사일이 계속 날라가는 경우
    GLCD_displayChar(missile_posX[i], missile_posY[i], ' '); //이전 위치 공백으로 덮어 쓰기
    missile_posX[i] += 6; //좌표 이동
    GLCD_displayChar(missile_posX[i], missile_posY[i], missile_shape);//LCD에 변경 좌표에
    if(missile_posX[i] >=320){ //미사일이 끝에 도착한경우
     isMissile[i] = 0; //미사일 제거
    }
   }
  }
  i++;
```

```
  }
}
//미사일 발사
void fire(){
 missile_posX[missile_num] = 24; //미사일 시작 x좌표 (고정)
 missile_posY[missile_num] = fighter_posY; //미사일 시작 y좌표 (비행기의 y좌표)
 isMissile[missile_num] = 1; //미사일 생성
 missile_num++; //미사일 개수 증가
 if(missile_num == missile_max) missile_num=0; //미사일 개수 유지
 missileMove(); //미사일 이동
}

//게임 종료
void gameOver(){
 GLCD_clear(Black); //검정색으로 LCD초기화
 GLCD_setBackColor(Black); //LCD 배경 검정색 초기화
 GLCD_setTextColor(White); //글자색 흰색으로 초기화
 if(isPassed == 0){ //목숨이 0일 경우
  GLCD_displayStringLn(Line1, "      GAME OVER      ");
  GLCD_displayStringLn(Line2, "  You Missed meteor ");
 }
 else{ //비행기가 부서진 경우
  GLCD_displayStringLn(Line1, "      GAME OVER   ");
  GLCD_displayStringLn(Line2, "    fighter Crashed");
 }
 GLCD_displayStringLn(Line3, "  Your Point : ");
 //포인트 출력
 if(point == 0){
  GLCD_displayChar(260, Line3, '0');
 }
 else{
  int x=260, y=Line3;
  while(point>0){
   char tmp = (char)(point%10+'0');
   GLCD_displayChar(x, y, tmp);
   x-=20;
   point = point/10;
  }
 }
 GLCD_displayStringLn(Line5, "  Push RESET Button ");
 while(1){ //리셋 버튼 누를때까지 대기
 }
}
```

```
//비행기 파괴
void fighterCrash(){
 if((meteor_posY+10 >= fighter_posY && meteor_posY -10<= fighter_posY) &&(meteor_posX
<= fighter_posX +20)){ //비행기 파괴 조건
  gameOver(); //게임 종료
 }
}
//장애물 이동
void movemeteor(){
 GLCD_displayChar(meteor_posX, meteor_posY, ' '); //이전 위치 공백으로 표기
 meteor_posX -= meteor_speed; //좌표 이동
 fighterCrash(); //비행기 파괴 검사
 GLCD_displayChar(meteor_posX, meteor_posY, meteor_shape); //변경 위치 LCD표기
 if(meteor_posX <= 1) { //장애물을 놓친 경우
  isPassed -= 1; //목숨 1감소
  GLCD_displayChar(200, Line9, (char)(isPassed+'0')); //목숨 수 LCD에 표기
  if(isPassed == 0){ //목숨이 끝난 경우
   gameOver();
  }
  GLCD_displayChar(meteor_posX, meteor_posY, ' ');
  createmeteor(); //새로운 장애물 생성
 }
}
//비행기 이동
void fighterMove(int key){
 int tmpX = fighter_posX; //이전 좌표
 int tmpY = fighter_posY;

 if(key == up) { //위로 이동
  if(fighter_posY == 0) fighter_posY = 0;
  else fighter_posY -= 4;
 }
 else if(key == down) { //아래로 이동
  if(fighter_posY == Line8) fighter_posY = Line8;
  else fighter_posY += 4;
 }
 else if(key == right){ //미사일 발사
  fire();
 }
 key = -1;
 GLCD_displayChar(tmpX, tmpY, ' ');
 GLCD_displayChar(fighter_posX, fighter_posY, fighter_shape); //새로운 좌표로 LCD표기
}
```

```
//게임 초기화
void gameInit(){
 fighter_posX = 0; //비행기 x좌표 초기화
 fighter_posY = 120; //비행기 y좌표 초기화
 meteor_posX = 300; //장애물 x좌표 초기화
 meteor_posY = 120; //장애물 y좌표 초기화
 missile_num = 0; //미사일 개수 0개로 초기화
 meteor_speed = 5; //초기 장애물 속도
 key = -1;
 isShoot = 0;
 isPassed = 3;  //목숨수 설정
 inGame = 0;
 point = 0; //점수 초기화
 fighter_shape = '>'; //각각 LCD표기 설정
 meteor_shape ='O';
 missile_shape ='-';

 GLCD_setBackColor(White); //배경색 초기화
 GLCD_setTextColor(Black); //글자색 초기화
 GLCD_displayChar(100, Line9, (char)(point+'0')); //포인트, 목숨 LCD초기 표기
 GLCD_displayChar(200, Line9, (char)(isPassed+'0'));
}
//조이스틱 조작
void joyStick (void) {
 int counter=0;
 uint32_t kbd_val;
 kbd_val = (LPC_GPIO1->FIOPIN >> 20) & KBD_MASK;
 key = -1;
 if ((kbd_val & KBD_UP)      == 0) key = up;
 if ((kbd_val & KBD_LEFT)    == 0) key = left;
 if ((kbd_val & KBD_RIGHT)   == 0) key = right;
 if ((kbd_val & KBD_DOWN)    == 0) key = down;

 while(counter<700000){ //잠깐의 텀 추가
  counter++;
 }
}
//게임 실행 함수
void runGame(){
 gameInit(); //게임 초기화
 inGame = 1; //게임 실행 표시
 while(inGame == 1){
  key = -1;      //이동 초기화
```

```
    joyStick();       //조이스틱 조작
    fighterMove(key); //비행기 이동
    missileMove(); //미사일 이동
    movemeteor(); //장애물 이동
  }
}

int main (void){
  GLCD_init(); //LCD  초기화
  GLCD_clear(White);
  runGame(); //게임 실행
}
```

7. 조원 역할

김진산

- 코드, ppt, 보고서 작성

최성우

- ppt 및 발표

정우

## 8. 참고 문헌 및 자료

http://bitly.kr/ZXcjrthWrQ

ARM Cortex-M3 NXP 응용 및 실습, HUINS

user_manual_LPC17xx

위키백과

https://m.blog.naver.com/PostView.nhn?blogId=sigsaly&logNo=220661007733&proxyReferer=https:%2F%2Fwww.google.com%2F