**2013학년도 2학기 마이크로프로세서 응용**

**텀 프로젝트 최종 보고서**



|  |  |
| --- | --- |
| **과목명** | **마이크로프로세서 응용** |
| **교수님** | **조인휘 교수님** |
| **전공** | **컴퓨터전공** |
| **학번** | **2012003738** |
| **이름** | **차정민** |

**보고서 목차**

1. **텀 프로젝트 개요**
2. **핵심 코드 설명**
   1. **Main 함수**
   2. **DisplayInMatrix 함수**
   3. **CreateObstacle/KillObstacle 함수**
   4. **DispInteger 함수**
   5. **Timer/Overflow Interrupt에 대한 루틴**
   6. **기타: 사용한 라이브러리에 대한 설명**
3. **한계점 및 개선 노력**
4. **텀 프로젝트 모든 소스 파일**
5. **텀 프로젝트를 마치며……**
6. **텀 프로젝트 개요**

제가 진행한 텀 프로젝트의 목표는 안드로이드 및 아이폰 등의 모바일 어플리케이션 게임 중 ‘미니언 러쉬’라고 하는 게임을 모방한 게임을 만드는 것이었습니다. ‘미니언 러쉬’ 게임은 3D그래픽 기반 게임이지만, 게임의 행동 패턴은 단순하였기 때문에 실습시간에 사용한 HBE-MCU 키트에서 구현할 수 있다고 생각하여 이번 텀 프로젝트의 목표로 결정하게 되었습니다.

‘미니언 러쉬’ 게임은 내 캐릭터가 점점 빠른 속도로 맵을 달리기 시작하며, 장애물들을 여러 가지 방법으로 피하면서 생존해나가는 게임입니다. 장애물의 종류에는 너무 커서 반드시 피해야 하는 장애물, 점프할 수 있는 장애물, 그리고 숙여야만 피할 수 있는 장애물이 있습니다. 장애물이 다가오는 모습과 장애물의 종류를 구분하여 피하는 방법을 달리 하면 되는 것이므로, 2차원 상에서도 그러한 모습을 구현만 한다면 실습에서 사용한 MCU 키트에서도 구현할 수 있을 것이라고 생각하였습니다.

텀 프로젝트를 진행하기 앞서 제안서 발표를 할 때에는 2Color Dot Matrix모듈을 이용해서 장애물 및 캐릭터의 위치를 표시하고, OLED 모듈을 이용해 다가오는 장애물의 형태를 파악하는 식의 게임을 구현하려고 하였으나, MCU키트에서 포트가 부족하다는 점, 그리고 게임이 덜 직관적이라는 단점을 개선하기 위해서 OLED 모듈을 사용하지 않고 구현하였습니다.

그리고, 보고서를 작성하는데 있어서, 최대한 조교님과 교수님께서 직관적으로 코드를 이해하시도록 돕기 위해서 코드에 주석을 다는 일에 더 신경을 많이 썼습니다.

1. **핵심 코드 설명**

**\* 보고서에 설명을 넣기보다 최대한 주석 코드에 코드에 대한 설명을 더 많이 담으려고 노력했습니다.**

1. main 함수

main 함수는 다음과 같이 크게 6가지 부분으로 나뉘어집니다.

**a) 전원 인가 후 작동해야 할 초기화 및 루틴**

**b) 메뉴 출력에 대한 구현**

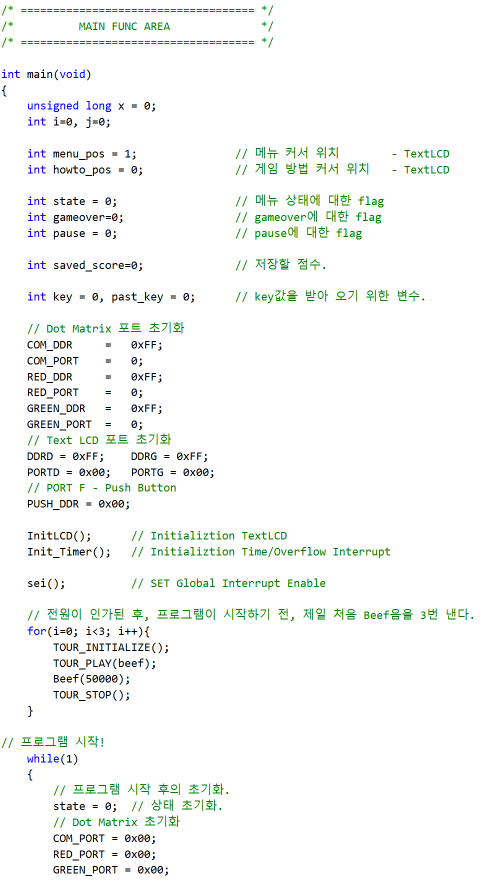
**c) 게임 실행 구현**

**d) 게임 설명 출력 구현**

**e) 기록 점수 출력 구현**

**f) 게임 종료**

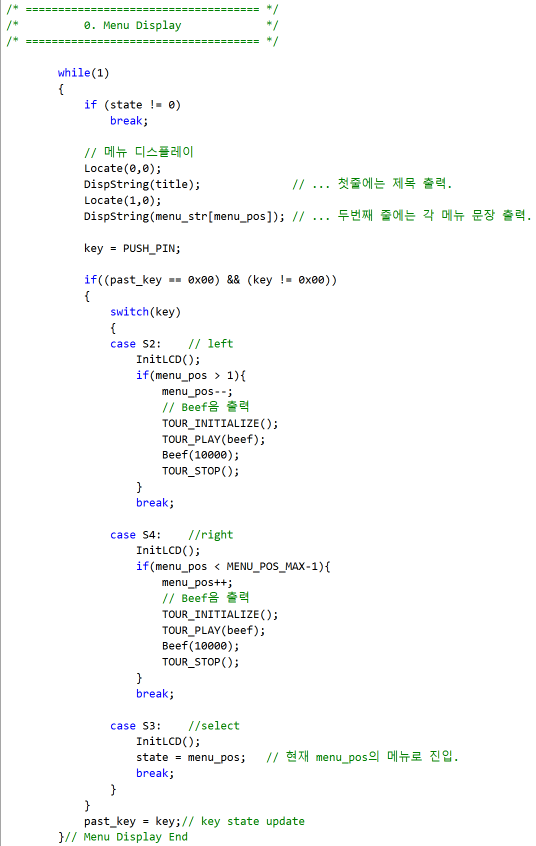
1. **전원 인가 후 작동해야 할 초기화 및 루틴**



* Figure 1 전원 인가 후 작동해야 할 초기화 및 루틴

Main 함수가 시작되면, 우선 게임에 필요한 여러 가지 자동변수들을 초기화 및 선언해주고, 사용할 포트들에 대해서 초기화합니다. 그리고, 피에조 모듈을 이용해서 세 번의 비프음을 낸 후에, while문에 진입하여 실질적인 프로그램이 시작되게 됩니다. 그 이후, state 및 Dot Matrix를 초기화하는데, 이것은 게임이 다 끝난 후에 다시 처음으로 돌아올 때에도 이들은 초기화 과정을 반드시 거쳐야 하기 때문입니다.

1. **메뉴 출력에 대한 구현**



* Figure 2 메뉴 출력에 대한 구현

위 코드는 프로그램이 시작되면 제일 처음 보게 될 TextLCD에서의 메뉴 출력 부분에 대한 구현 소스입니다. 제일 처음 state=0으로 초기화하였으므로 프로그램 시작 시 반드시 이 루틴을 진행하게 됩니다. TextLCD 첫번째 줄에는 게임의 제목이, 두번째 줄에는 메뉴 선택에 대한 문장들이 출력됩니다. 버튼을 누를 때마다 피에조를 이용해서 Beef음을 출력하도록 하였습니다. 그리고, 선택 버튼을 누르면 TextLCD를 초기화하고, state값을 바꾸어 이 while문을 탈출합니다. 그리고 그 state값에 맞추어 알맞은 while문으로 이동하도록 구현하였습니다.

1. **게임 실행 구현**

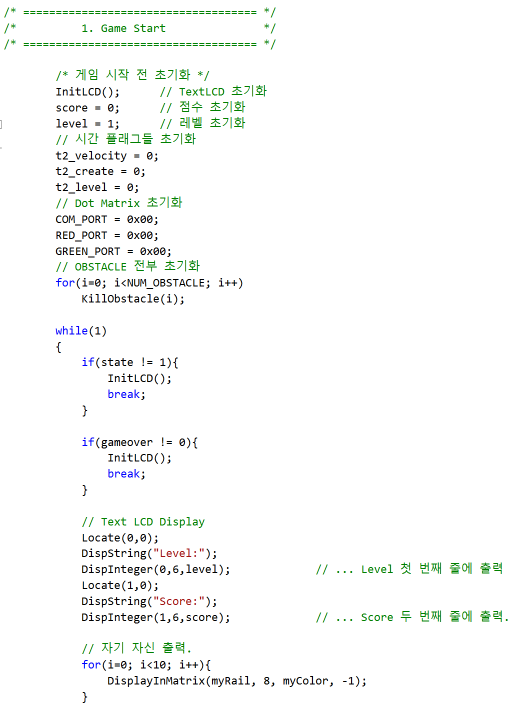


Figure 3 게임 실행 구현 – 1 초기화 및 TextLCD 출력

이제, 진짜 실질적인 게임이 진행될 때의 코드를 설명하겠습니다. 게임 시작 전 초기화 되어야 하는 부분을 전부 초기화합니다. (점수, 레벨, 시간 플래그, DotMatrix등) 그리고 장애물 구조체 배열들을 전부 초기화(Kill)시키고, 게임의 while문에 진입합니다. 앞에서 Game Start 메뉴를 선택하였다면 state값이 1이 되었을 것이므로 이 while문을 빠져나가지 않고 진행합니다. TextLCD에는 현재 레벨과 스코어가 출력이 됩니다. 그리고 자기 자신을 출력하는 함수를 호출하였는데, 10번을 호출한 이유는 캐릭터를 좀 더 선명하게 출력하기 위함입니다.



Figure 4 게임 실행 구현 – 2 장애물 처리 및 점프 구현부

위 소스 코드는 장애물의 종류에 따라서 장애물을 만났을 때 게임오버 flag를 on시켜서 게임오버 루틴을 진행하도록 하는 부분입니다. 그 밑에 점프 구현에 대한 소스 코드가 있습니다. 점프 키를 누르고 있는 중에는 jump flag를 on시켜서 jump인 상태가 되고, 캐릭터의 색이 오렌지가 되도록 하였습니다.

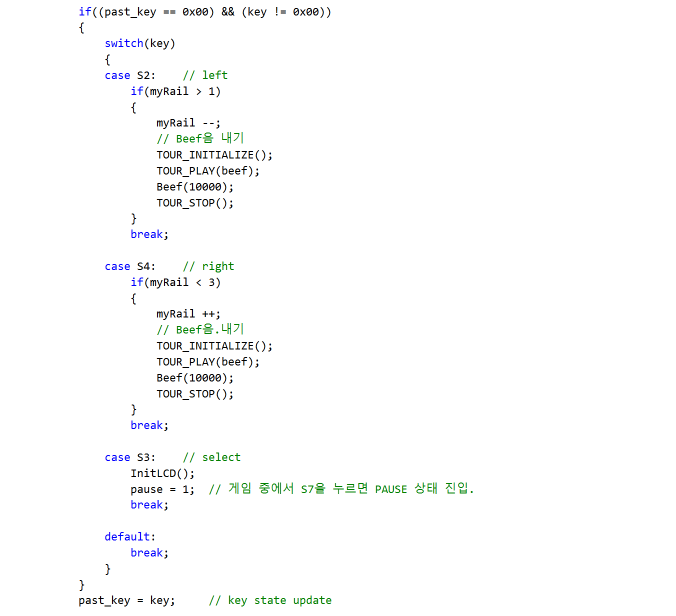


Figure 5 게임 실행 구현 – 3 키 입력에 대한 처리

점프 키를 제외하고는 지속적으로 눌러야 하는 키가 없으므로, past\_key를 사용하여 한 번의 클릭만을 인지하도록 하였습니다. 게임 실행 중에도 캐릭터를 왼쪽 오른쪽으로 움직이는 경우, Beef음을 출력하도록 하였습니다. 또, Select/Pause버튼을 누르게 되면 pause flag를 on 시켜서 PAUSE 상태로 진입하도록 하였습니다.

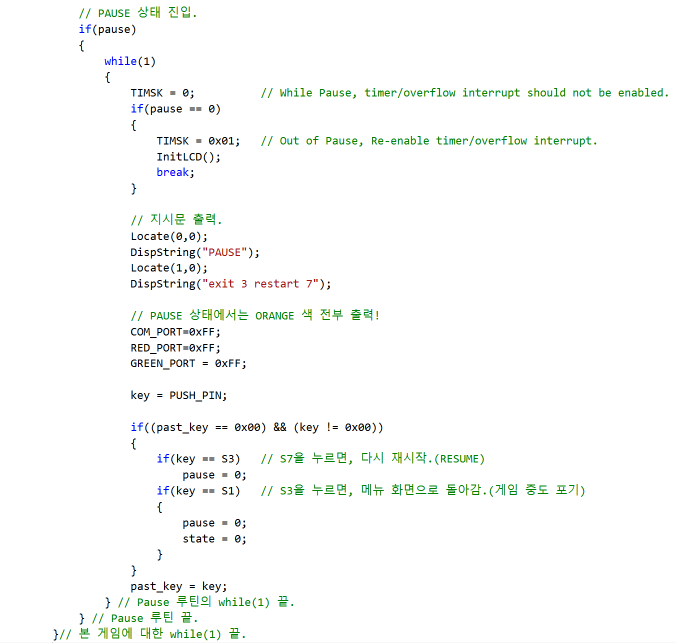


Figure 6 게임 실행 구현 – 4 PAUSE 상태에 대한 루틴

PAUSE 상태에 진입한 경우, Timer/Overflow 인터럽트를 꺼서, 더 이상 게임 상의 실행시간이 진행되지 않도록 하였습니다. (인터럽트가 계속해서 돈다면, PAUSE인 상태에서도 계속해서 내려오던 장애물이 계속해서 내려 올 것이며, 레벨이나 점수가 지속적으로 계속 증가하는 문제가 생길 것입니다.) 또, Text LCD에서는 PAUSE 상태를 표시하는 문장과 함께 나가는 방법 및 되돌아가는 방법에 대한 문장을 표시하였습니다. 또, Dot Matrix의 모든 칸을 전부 오렌지 색으로 출력하여 PAUSE 상태를 조금 더 시각적으로 보여주었습니다.

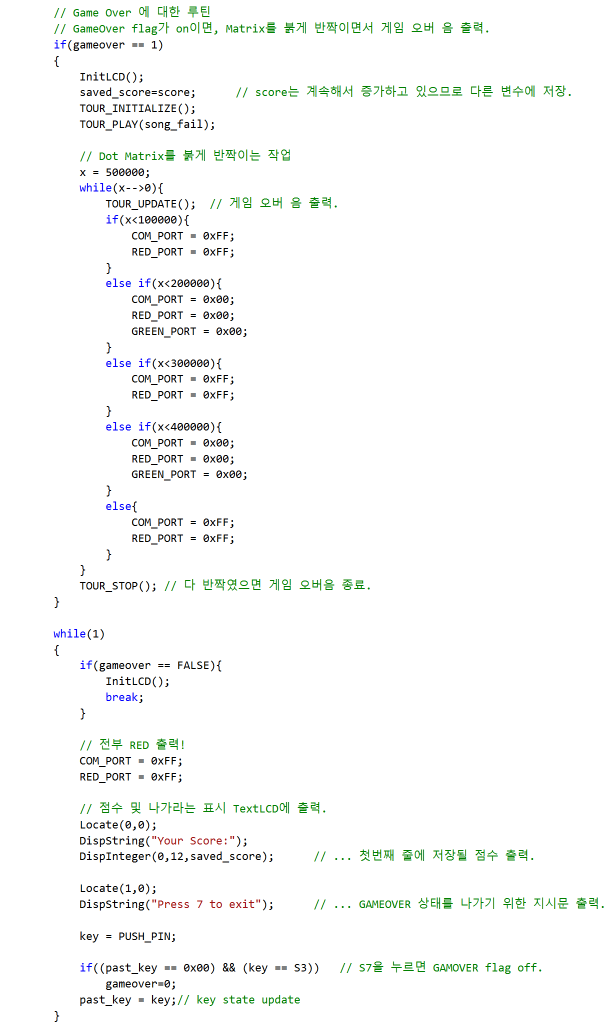


Figure 7 게임 실행 구현 – 5 게임 오버 루틴

게임 오버한 경우, Dot Matrix를 붉은 색으로 세 번 정도 반짝이며 게임 오버 노래가 나오며, Text LCD에서는 이번 게임의 score가 출력됩니다.

1. **게임 설명 출력 구현**

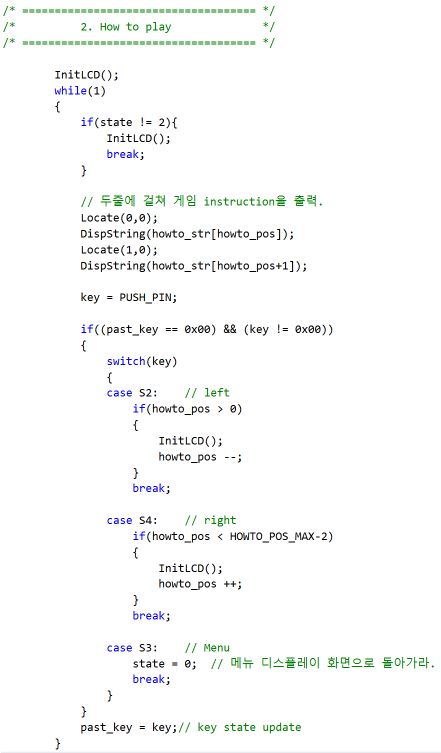


Figure 8 게임 설명 출력 구현

위 소스는 두번째 메뉴인 ‘2. How to’에 진입하였을 때 TextLCD에서 게임 설명을 출력할 수 있도록 짠 부분입니다.

1. **기록 점수 출력 구현**

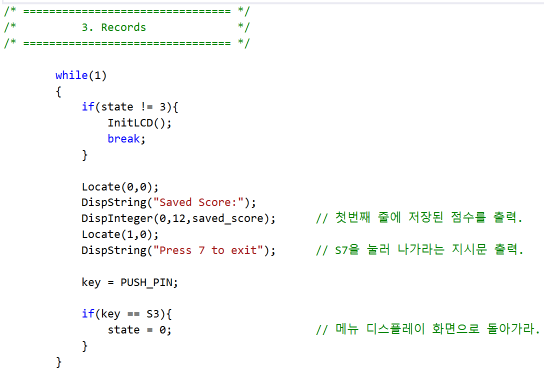


Figure 9 기록 점수 출력 구현

위 소스 코드는 3번째 메뉴인 ‘3.Records’에 진입하였을 때, 기록 점수를 출력하는 구현부입니다. 가장 마지막에 기록된 점수를 출력해줍니다.

1. 게임 종료

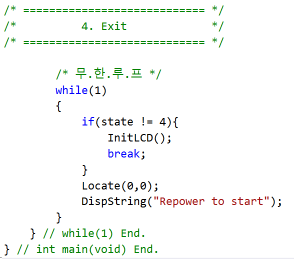


Figure 10 게임 종료

위 소스 코드는 4번째 메뉴인 ‘4.Exit’에 진입하였을 때, 즉, 게임 종료 메뉴의 구현부입니다. 이 메뉴 상태에서는 무한루프를 돌며, 반드시 전원을 재인가 하여야 빠져 나갈 수 있습니다.

1. DisplayInMatrix 함수

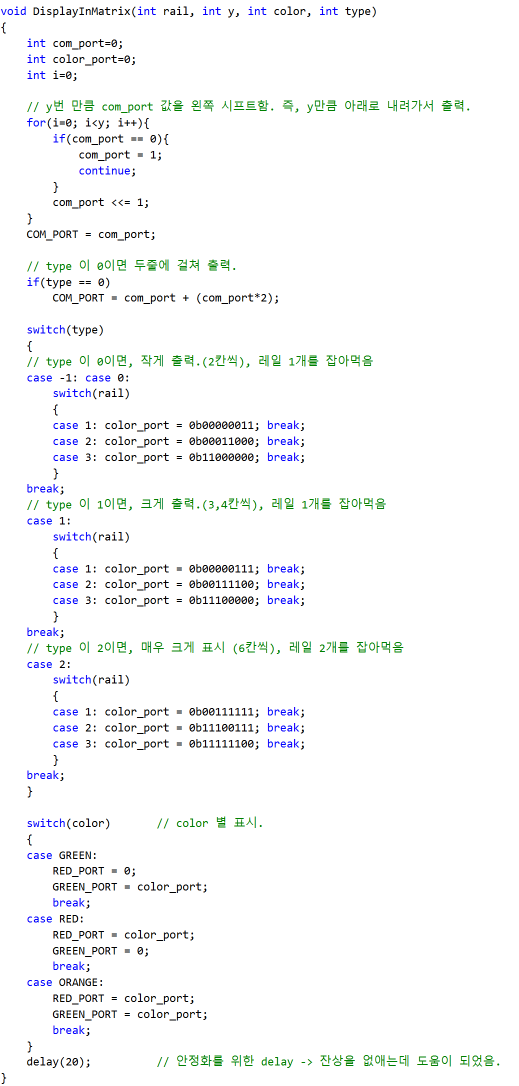


Figure 11 DisplayInMatrix 함수 구현부

장애물 및 캐릭터가 어느 레일, 어느 위치에 있고, 어떤 색과 타입인지에 따라 DotMatrix에 출력해주는 함수 루틴입니다.

1. CreateObstacle/KillObstacle 함수

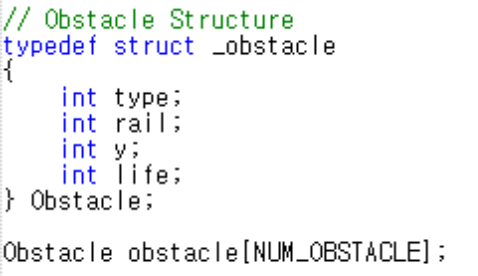


Figure 12 장애물 구조체 선언부

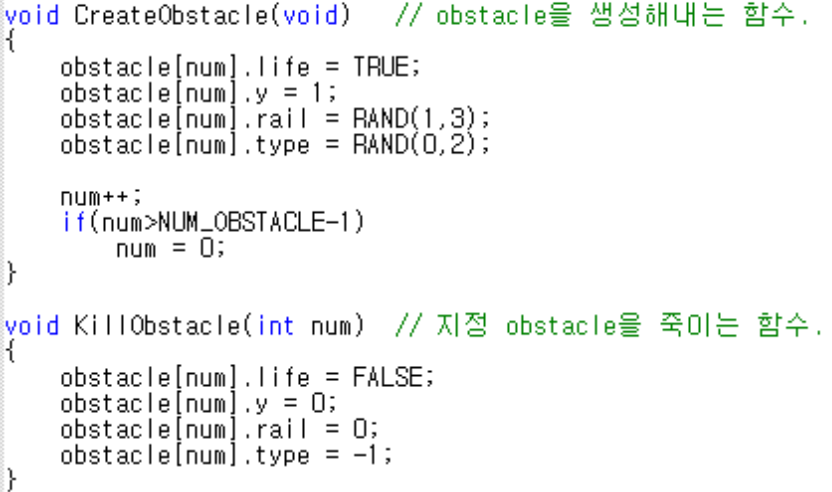
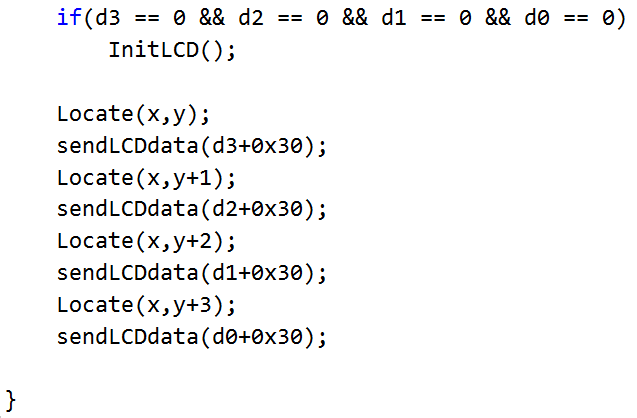
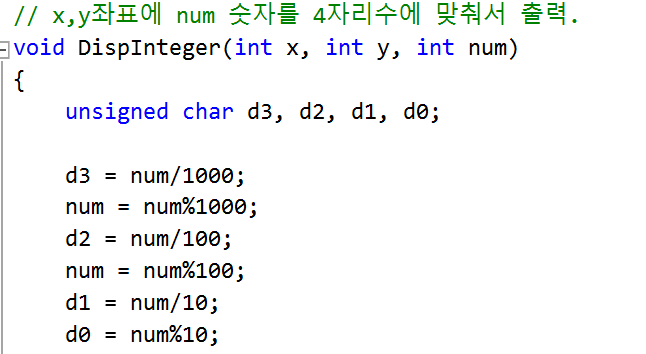


Figure 13 장애물을 만들고 죽이는 함수 구현부

Figure 12와 같이 장애물 구조체를 선언하였고, 장애물의 life를 TRUE/FALSE로 처리하여, 죽었는지 살았는지를 판단하였습니다. 메인함수나 타이머 오버플로우 인터럽트 루틴 속에서 장애물의 life를 판단한 후에 출력하거나 장애물에 대한 처리(게임오버, 혹은 장애물이 계속해서 내려오거나 생성하는 등의 처리)를 진행하도록 하였습니다.

1. DispInteger 함수



매개변수로 입력 받은 숫자를 Text LCD모듈에 4자리에 맞추어 출력해주는 함수입니다.

1. Timer/Overflow Interrupt에 대한 루틴

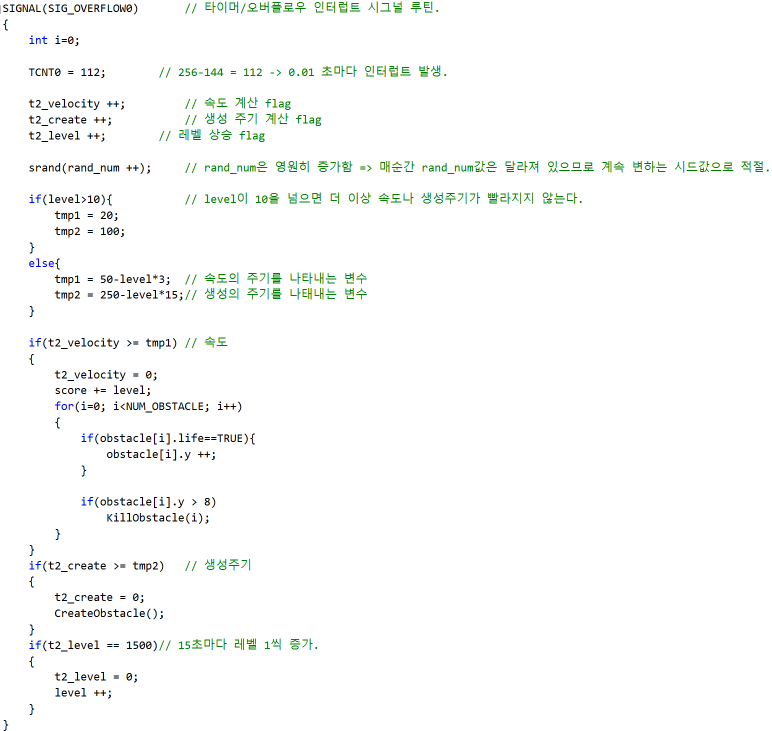


Figure 14 Timer/Overflow Interrupt에 대한 루틴

타이머/오버플로우 인터럽트에서는 시간 플래그 들에 대한 처리를 담았습니다. 게임 속 난이도의 상승을 위해, 레벨이 올라가는 처리 및 장애물이 생성되고, 내려오는 속도에 대한 조절이 필요했습니다. 이러한 구현들을 전부 인터럽트 실행부에 집어넣었습니다. 시간 플래그를 총 3개를 선언해두어, 장애물이 내려오는 속도를 관리하고, 생성 주기를 관리하고, 레벨이 오르는 주기를 관리하는 시간 플래그를 선언해두었습니다.

1. 기타: 사용한 라이브러리에 대한 개요

특별히 사용한 라이브러리는 TOUR 라이브러리입니다. 이 라이브러리는 피에조 모듈을 이용해서 비프음 및 노래를 출력하기 위해서 사용하였습니다. TOUR Library는 **TOUR\_INITIALIZE() -> TOUR\_PLAY(song\_name) -> TOUR\_UPDATE()의 반복 -> TOUR\_STOP()** 과 같은 방식으로 사용할 수 있습니다. 저는 짧은 음을 내기 위해서, Beef라는 함수를 따로 만들어, Beef함수가 호출되는 특정 시간 동안, TOUR\_UPDATE()가 계속해서 반복 호출되어 적당히 짧은 특정 시간동안의 소리 출력으로 비프음출력을 구현하였습니다. 또한, 게임 오버 시에는 계속해서 TOUR\_UPDATE를 해주면서, DotMatrix 포트의 값들을 주기적으로 바꾸어줌으로써 노래도 나오면서 DotMatrix가 깜빡일 수 있도록 해주었습니다.

1. **한계점 및 개선 노력**

프로젝트를 진행하면서 어려웠던 점들이 몇 가지 있었습니다.

**첫번째, DotMatrix가 선명하게 보이지 않으며 번인 현상이 발생하는 문제**가 있었습니다. DotMatrix 모듈에서 이미 있던 장애물과 새롭게 생긴 장애물의 선명도가 눈에 띄게 차이가 나며, 장애물이 없는 곳인데도 약간의 흐릿한 led 불빛이 들어오는 잔상 현상, 즉, 번인 현상이 발생하였습니다. 저는 문제의 발생 원인이 Display 되는 부분과 그 외 게임 상황에 대한 처리가 같은 while문 안에 있다 보니, 디스플레이 구현 부에서 소비되는 시간보다 게임 상황 처리 부분에서 더 많이 시간이 소비됨으로써, DotMatri 디스플레이가 흐릿하게 나오고 잔상이 생기는 것이라고 판단하였습니다. 이 문제를 해결하기 위해1)DisplayInMatrix 함수 자체 내부에 약간의 딜레이를 주었으며, 이것으로도 전부 해결되지 않자, 메인 함수 내에서 2)DisplayInMatrix 함수를 호출할 때, 여러 번(약 10번 정도) 호출을 하도록 하였습니다.

**두 번째,** 시간이 지남에 따라 난이도를 조절하기 위해서 맞추는 변수들 값에 따라서 **특정 레벨에서 게임이 진행되지 않고 뻗어버리는 문제**가 있었습니다. 타이머/오버플로우 인터럽트 실행부에서 과한 함수 호출로 인해 시간 플래그가 올라가는 것을 제대로 잡아내지 못하는 것이 문제의 원인이었습니다. 코드의 수준을 높이기 위해서는 인터럽트 실행부에서 함수를 바로 호출하지 말고 플래그 변수들을 이용해서 메인 함수에서 전부 호출할 수 있도록 바꿀 수도 있었지만, 저는 시간 플래그의 값들을 체크할 때, 등호 연산을 부등호 연산으로 바꿈으로써 문제를 해결하였습니다. 이러한 해결 방법은 정말 짧은 순간의 함수 처리에 있어서 정확한 처리가 힘들수도 있다는 새로운 문제를 만들지만, 제가 만든 게임 속에서 이러한 상황은 발생할 확률이 거의 적기 떄문에 이러한 방식으로 문제를 해결하였습니다.

**세 번째, Text LCD에서 정수를 출력해야 할 때** 약간의 난관을 겪었습니다. 실습시간에 사용하였던 함수들은 모두 char 형식의 변수들을 출력할 때 사용할 수 있는 함수들이어서 바로 정수를 출력할 수 없었습니다. 정수를 문자 형식으로 변환해주는 atoi 함수가 제대로 사용이 되지 않아, 입력받은 숫자를 4자리 숫자에 맞추어 출력하도록 함수를 새로 구현하였습니다.

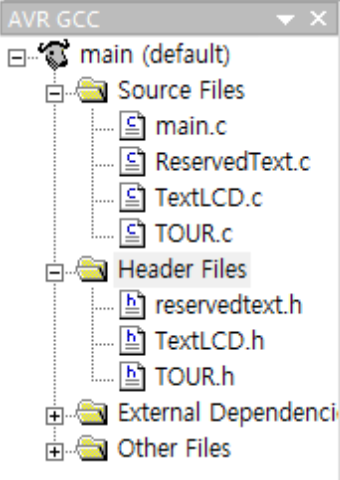
**네 번째**, 사용한 피에조 모듈의 **TOUR라이브러리의 사용성에 대한 한계**에 대한 처리도 어려웠던 점 중 하나였습니다. TOUR 라이브러리는 피에조 모듈의 사용을 위해서 TOUR\_UPDATE()함수의 반복적인 호출을 요구로 하는데, 특정 시간만 소리를 내기 위해서는 정해진 반복횟수를 가진 반복문을 이용해야 했습니다. 그런데, 무한루프를 돌릴 때 나는 소리와는 다르게 약간의 딜레이가 겹쳐서 소리가 재생되는 문제가 발생하였습니다. 이는 반복문이 진행됨에 따라 특정 변수를 ++ 혹은 -- 연산을 진행하는데 시간이 소모되기 때문입니다. 따라서, 저는 원래 소리의 속도를 2배 이상 더욱 빠르게 하고, 특정 시간 동안 소리를 낼 수 있도록 Beef 함수를 구현하였습니다.

**다섯 번째**, **랜덤함수에 시드 값을 주는 방법**에서도 고난을 겪었습니다. gcc등 범용적 컴파일러로 컴파일하는 일반적인 환경에서 C 프로그래밍을 할 때, 랜덤함수를 구현하는 방법은 랜덤함수의 시드 값에 현재 시간을 계속해서 집어넣어 줌으로서 매 시간마다 다른 시드 값을 통해서 실질적인 의미의 랜덤함수를 구현하였습니다. 하지만, 실습 환경에서는 time.h 헤더파일을 제대로 사용할 수 없었기 때문에 매 시간마다 달라지는 값을 찾아야만 했습니다. 그래서 저는 전역변수로 rand\_num이라고 하는 변수를 잡아서, 전원이 인가되는 순간부터 계속해서 증가하도록 하였고 이를 랜덤함수의 시드 값으로 주었습니다. 사람이 게임을 시작하는 데 있어서 완전히 똑같은 시간이 지난 후에 게임을 시작할 수는 없기 때문에, 매 게임이 진행될 때마다 rand\_num값은 다른 값인 채로 게임이 진행될 것이라고 가정하였습니다. 실제로도 게임을 여러 번 진행해보았지만 랜덤 함수에 규칙적인 모습은 전혀 보이지 않았습니다.

1. **텀프로젝트 모든 소스 파일**

**\* 피에조 모듈을 위해서 사용한 TOUR Library는 소스 코드가 너무 방대해서 보고서의 분량이 지나치게 많아지는 점을 감안해서 첨부만 하였습니다.**

**\* 핵심 함수들의 대부분은 위에서 언급하였으므로 위와 같은 이유로 언급되지 않은 부분만 담았습니다.**

**소스 파일의 개요는 다음과 같습니다.**

소스 파일은 main.c, ReservedText.c, TextLCD.c, TOUR.c가 있고, 헤더파일에는 reservedtext.h, TextLCD.h, TOUR.h가 있습니다.

main.c 에서는 main() 함수와 인터럽트 구현부, DisplayInMatrix, 장애물에 대한 처리, 게임 상황에 대한 처리 등이 담긴 소스 파일입니다.

ReservedText.c/h 에서는 시스템에서 예약되어있는 문장들을 저장하는 데 사용하였습니다. 가령, 메뉴 문장이나, 게임 설명에 대한 문장들이 이 곳에 저장되어 있습니다.

TextLCD.c/h 에서는 실습시간에 TextLCD 모듈을 위해서 사용했던 함수들과 제가 직접 구현한 DispInteger함수가 있습니다.

TOUR.c/h 에서는 피에조 모듈을 위해 사용한 TOUR 라이브러리 함수들이 들어있습니다.

* main.c (언급되지 않은 부분만)

|  |
| --- |
| /\* ===========================================  전체 포트 배선 사용 개요  2 Color Dot Matrix - PORTE / PORTA / PORTC  TextLCD - PORTD / PORTG  PushButton - PORTF  PF0/PF1 - BT2/BT3, PF2/PF3 - BT6/BT7  PIEZO - PB 5  ============================================== \*/  /\* ================================ \*/  /\* HEADER FILES \*/  /\* ================================ \*/  #include <avr/io.h>  #include <avr/interrupt.h>  #include <stdlib.h>  #include "TOUR.h"  #include "ReservedText.h"  #include "TextLCD.h"  /\* ==================================== \*/  /\* DEFINITION AREA \*/  /\* ==================================== \*/  #define TRUE 1  #define FALSE 0  #define NUM\_OBSTACLE 7  /\* 2 Dot Color Matrix Definition \*/  #define COM\_PORT PORTE  #define COM\_DDR DDRE  #define RED\_PORT PORTA  #define RED\_DDR DDRA  #define GREEN\_PORT PORTC  #define GREEN\_DDR DDRC  /\* Push Button - PORTF \*/  #define PUSH\_DDR DDRF  #define PUSH\_PORT PORTF  #define PUSH\_PIN PINF  #define S1 0x01  #define S2 0x02  #define S3 0x04  #define S4 0x08  #define S5 0x10  #define S6 0x20  #define S7 0x40  #define S8 0x80  #define PUSH\_MASK 0b11001100  /\* ==================================== \*/  /\* VARIABLE AREA \*/  /\* ==================================== \*/  volatile unsigned int t2\_velocity, t2\_create, t2\_level; // 타이머 인터럽트에서의 시간 flag들  unsigned char beef[] = { 'T', 255,  'O', 6, 'C', 64,  0 };  unsigned char song\_fail[] = { 'T', 255,  'O', 6, 'I', 2, 'R', 2, 'E', 2, 'R', 2, 'E', 2, 'R', 2, 'E', 2, 'R', 2, 'O', 5, 'I', 2, 'R', 2, 'E', 2, 'R', 2, 'E', 2, 'R', 2, 'E', 2, 'R', 2,  'O', 4, 'I', 2, 'R', 2, 'E', 2, 'R', 2, 'I', 2, 'R', 2, 'E', 2, 'R', 2, 'O', 3, 'G', 6, 'R', 2, 'O', 2, 'G', 6, 'R', 2,  'M',  'O', 3, 'C', 2, 'R', 2, 'O', 4, 'C', 2, 'R', 2, 'O', 3, 'C', 2, 'R', 2, 'O', 4, 'C', 2, 'R', 2, 'O', 3, 'E', 2, 'R', 2, 'O', 4, 'E', 2, 'R', 2, 'O', 3, 'E', 2, 'R', 2, 'O', 4, 'E', 2, 'R', 2,  'O', 3, 'F', 2, 'R', 2, 'O', 4, 'F', 2, 'R', 2, 'O', 3, 'F', 2, 'R', 2, 'G', 2, 'R', 10, 'O', 2, 'G', 6, 'R', 2,  'O', 3, 'C', 2, 'R', 2, 'O', 4, 'C', 2, 'R', 2, 'O', 3, 'C', 2, 'R', 2, 'O', 4, 'C', 2, 'R', 2, 'O', 3, 'E', 2, 'R', 2, 'O', 4, 'E', 2, 'R', 2, 'O', 3, 'E', 2, 'R', 2, 'O', 4, 'E', 2, 'R', 2,  'O', 3, 'F', 2, 'R', 2, 'O', 4, 'F', 2, 'R', 2, 'O', 3, 'F', 2, 'R', 2, 'G', 2, 'R', 18,  'O', 3, 'C', 2, 'R', 2, 'O', 4, 'C', 2, 'R', 2, 'O', 3, 'C', 2, 'R', 2, 'O', 4, 'C', 2, 'R', 2, 'O', 3, 'E', 2, 'R', 2, 'O', 4, 'E', 2, 'R', 2, 'O', 3, 'E', 2, 'R', 2, 'O', 4, 'E', 2, 'R', 2,  'O', 3, 'F', 2, 'R', 2, 'O', 4, 'F', 2, 'R', 2, 'O', 3, 'F', 2, 'R', 2, 'G', 2, 'R', 10, 'O', 2, 'G', 6, 'R', 2,  'O', 3, 'A', 2, 'R', 10, 'D', 7, 'R', 1, 'D', 2, 'R', 2, 'D', 2, 'R', 2, 'D', 2, 'R', 2,  'G', 2, 'R', 10, 'C', 7, 'R', 1, 'C', 2, 'R', 2, 'C', 2, 'R', 2, 'C', 2, 'R', 2,  'F', 2, 'R', 10, 'D', 7, 'R', 1, 'D', 4, 'G', 3, 'R', 1, 'G', 4,  'C', 2, 'R', 10, 'O', 2, 'H', 7, 'R', 1, 'I', 2, 'R', 2,  'C', 2, 'R', 14,  'O', 3, 'F', 7, 'R', 1, 'O', 4, 'C', 4, 'F', 4, 'B', 4, 'O', 5, 'D', 4, 'C', 4, 'O', 4, 'A', 4,  'O', 3, 'E', 7, 'R', 1, 'O', 4, 'C', 4, 'E', 4, 'A', 4, 'O', 5, 'C', 4, 'O', 4, 'B', 4, 'G', 4,  'O', 3, 'D', 7, 'R', 1, 'A', 4, 'O', 4, 'D', 4, 'K', 4, 'B', 4, 'A', 4, 'F', 4,  'O', 3, 'C', 7, 'R', 1, 'G', 4, 'O', 4, 'C', 4, 'O', 3, 'D', 7, 'R', 1, 'E', 4, 'R', 4,  'F', 7, 'R', 1, 'O', 4, 'C', 4, 'F', 4, 'B', 4, 'O', 5, 'D', 4, 'C', 4, 'O', 4, 'A', 4,  'O', 3, 'E', 7, 'R', 1, 'O', 4, 'C', 4, 'E', 4, 'A', 4, 'O', 5, 'C', 4, 'O', 4, 'B', 4, 'G', 4,  'O', 3, 'D', 7, 'R', 1, 'A', 4, 'O', 4, 'D', 4, 'R', 16,    'O', 3, 'G', 7, 'R', 9, 'O', 2, 'G', 4, 'R', 12,  'O', 3, 'G', 7, 'R', 1, 'O', 2, 'G', 7, 'R', 1, 'G', 7, 'R', 1, 'B', 7, 'R', 1,  0 }; |

이 뒤로는 앞에서 언급한 순서대로 소스 코드가 있습니다.

* ReservedText.c/ reservedtext.h

|  |
| --- |
| **< ReservedText.c >**  #include "ReservedText.h"  // Game Title  char title[16] = "< Minion Rush! >";  // Sentences for menu  char menu\_str[MENU\_POS\_MAX][16] = {  " ",  "1.Game Start ->",  "<- 2.How to ->",  "<- 3.Records ->",  "<- 4.Exit"  };  // Sentences for instructions (how to play)  char howto\_str[HOWTO\_POS\_MAX][16] = {  "<Push Button>",  "left 4, right 8",  "up 3 Menu 7",  "<Obstacles>",  "1 line - jump x",  "2 line - jump o",  "press 7 to exit!"  }; |
| **< ReservedText.h >**  #ifndef \_\_RESERVED\_TEXT\_H\_\_  #define \_\_RESERVED\_TEXT\_H\_\_  /\*  \* There are sentences reserved in the game system,  \* including title, menu, instructions, etc.  \*/  #define MENU\_POS\_MAX 5  #define HOWTO\_POS\_MAX 7  char title[16];  char menu\_str[MENU\_POS\_MAX][16];  char howto\_str[HOWTO\_POS\_MAX][16];  #endif |

* TextLCD.c/TextLCD.h

|  |
| --- |
| **<TextLCD.c>**  #include <avr/io.h>  #include <stdlib.h>  #include "TextLCD.h"  void delay(unsigned long x)  {  while(x--);  }  void EnablePulse(void)  {  // ------  // 펄스 만들기  // ------  LCD\_CTRL |= LCD\_E; // E = 1로 만든다. 초기값(이전 값)은 0이었어야 한다.  asm("NOP"); // 아무것도 하지 않는 어셈블리 명령어.  // 1싸이클의 아주 짧은 시간 지연을 준다.  LCD\_CTRL &= ~LCD\_E; // 다시 E = 0으로 만든다.  }  void sendLCDcommand(unsigned char command)  {  LCD\_CTRL &= ~LCD\_RS; // 명령을 전달할 때 RS = 0  LCD\_DATA = command; // 전송할 명령 code 값을 data bus에 실어 놓는다.  EnablePulse(); // enable pulse 를 내보낸다. 이 시점에서 명령이 전송된다.  delay(20); // 전송된 명령을 LCD가 받아들여 처리하는데 걸리는 시간이 있음.  }  void InitLCD(void)  {  delay(20000);  LCD\_CTRL &= ~LCD\_RW; // LCD에 명령이나 자료를 전송할 때 항상 RW=0  sendLCDcommand(0x38); // function 설정 8bit data를 한 번에 처리.  // 2 line, 5\*7 dot.  sendLCDcommand(0x0C); // Display를 ON 시킨다. 이걸 안하면 글자 표시 안됨.  sendLCDcommand(0x01); // Clear Screen  delay(1000); // 1.53 ms 이상 되어야 함. -> 계산상 2 ms 이상으로 예측.  // Clear Screen 명령은 실행시간이 길기 때문에 충분히 기다려야 함.  }  // x,y좌표에 num 숫자를 4자리수에 맞춰서 출력.  void DispInteger(int x, int y, int num)  {  unsigned char d3, d2, d1, d0;  d3 = num/1000;  num = num%1000;  d2 = num/100;  num = num%100;  d1 = num/10;  d0 = num%10;  if(d3 == 0 && d2 == 0 && d1 == 0 && d0 == 0)  InitLCD();  Locate(x,y);  sendLCDdata(d3+0x30);  Locate(x,y+1);  sendLCDdata(d2+0x30);  Locate(x,y+2);  sendLCDdata(d1+0x30);  Locate(x,y+3);  sendLCDdata(d0+0x30);    }  void sendLCDdata(unsigned char data)  {  LCD\_CTRL |= LCD\_RS; // data를 보내기 위해 RS=1 만든다.  LCD\_DATA = data; // 전송할 문자 code값을 data bus에 실어 놓는다.  EnablePulse(); // enable pulse 를 내보낸다. 이 시점에서 명령이 전송된다.  delay(20); // data 전송 후 LCD가 처리하는 데 시간이 걸린다.  }  void DispString(char \* str)  {  while(\*str)  {  sendLCDdata(\*str);  str ++;  }  }  void Locate(int x, int y)  {  unsigned char RamAddr;    if(x==0)  RamAddr = 0x80 + y; // line1 D.D.RAM address  else  RamAddr = 0xC0 + y; // line2 D.D.RAM address  sendLCDcommand(RamAddr);  } |
| **<TextLCD.h>**  #ifndef \_\_TEXT\_LCD\_H\_\_  #define \_\_TEXT\_LCD\_H\_\_  /\*  \* TextLCD를 위한 포트 배선  1) PB0~7 을 TextLCD 의 D0~D7 에 연결한다.  2) PORTD는 다음과 같이 연결한다.  PD7 PD6 PD5 PD4 PD3 PD2 PD1 PD0  \* \* \* E \* \* RW RS  \* (!)나머지 배선하지 않은 PORT D의 포트들은 Push Button에 배선.  \*/  #define LCD\_CTRL PORTG  #define LCD\_DATA PORTD  #define LCD\_RS 0x01 // PD0  #define LCD\_RW 0x02 // PD1  #define LCD\_E 0x04 // PD4  #define TRUE 1  #define FALSE 0  /\* ======================================== \*/  /\* FUNCTION PROTOTYPE - for TextLCD \*/  /\* ======================================== \*/  void delay(unsigned long x);  void EnablePusle(void);  void InitLCD(void);  void sendLCDcommand(unsigned char command);  void sendLCDdata(unsigned char data);  void DispString(char \* str);  void DispInteger(int x, int y, int num);  void Locate(int x, int y);  #endif |

1. **텀 프로젝트를 마치며……**

이번 마이크로프로세서 응용 수업시간에 얻은 것 중에 가장 큰 경험은 MCU키트를 개발해보면서, 특정 MCU에 맞추어 개발해보는 경험이 아닐까 싶습니다. 인터넷을 통해 ATmega128에 대한 문서를 찾아가 보면서 MCU의 I/O 포트들이 어떠한 의미를 가지고 있고, 특정 변수들의 비트가 어떤 식으로 이루어져 있는지 의미를 되새겨 가면서 프로그래밍을 하는 경험은 2학년 1학기때까지만 해도 느끼기 힘든 경험이었습니다. 이전에는 추상적인 개념을 따져가며 프로그래밍을 했다고 한다면, 이번 마이크로프로세서 응용 수업에서는 실제로 실행시킬 기기에 대한 이해를 요구했기 때문에 프로그래밍의 개념이 더욱 현실적으로 다가오게 되었습니다. 제가 앞으로 어떤 공부를 해 내갈지는 아직 결정되지 않았지만, 만약에 MCU, 임베디드 개발을 하더라도 자신감 있게 해 나갈 수 있을 것 같습니다.