마이크로프로세서
FINAL PROJECT REPORT EXPLANATION

TEAM	4조
이름	정용의
	유승민
	전유성

```
#include <Arduino.h>
#include <MsTimer2.h>

// LED 핀 번호 정의
#define LED_NO_1  0
#define LED_NO_2  1
#define LED_NO_3  2
#define LED_NO_4  3
#define LED_NO_5  4
#define LED_NO_6  5
#define LED_NO_7  6
#define LED_NO_8  7
#define LED_NO_9  14
#define LED_NO_10  15
#define LED_NO_11  16
#define LED_NO_12  17
#define LED_NO_13  18
#define LED_NO_14  19
#define LED_NO_15  20
#define LED_NO_16  21

// FND 핀 정의 (디지털 핀 사용)
#define FND1  8
#define FND2  9
#define FND3  10
#define FND4  11
#define FND5  12
#define FND6  13
#define FND7  AREF
#define FND8  GND
#define FND_A  38
#define FND_B  39
#define FND_C  40
#define FND_D  41
#define FND_E  42
#define FND_F  43
#define FND_G  44
#define FND_DP 45

#define A_BIT  0x01
#define B_BIT  0x02
#define C_BIT  0x04
#define D_BIT  0x08
#define E_BIT  0x10
#define F_BIT  0x20
#define G_BIT  0x40
#define DP_BIT 0x80

#define FND_1_SEL  0x01
#define FND_2_SEL  0x02
#define FND_3_SEL  0x04
#define FND_4_SEL  0x08
#define FND_5_SEL  0x10
#define FND_6_SEL  0x20

#define MAX_FND_NUM 6
#define MAX_FND  8
```

먼저 라이브러리의 경우 타이머 인터럽트를 사용하기 위해 MsTimer2 라이브러리를 사용하였습니다. LED의 경우 LED의 핀 번호 LED_NO_1부터 LED_NO_16까지 16개의 LED가 사용됩니다. 다음으로는 FND(7_segment_display) 핀과 관련된 정의입니다. FND1부터 FND6는 FND의 선택 핀을 나타내고, FND_A부터 FND_DP는 각 segment를 나타내는 핀입니다. A_BIT부터 DP_BIT는 각 각의 segment 비트로 정의합니다. FND_1_SEL부터 FND_6_SEL은 각 FND를 선택하기 위한 비트 마스크입니다.

```
61    const byte LedPinTable[16] = {
62      LED_NO_1, LED_NO_2, LED_NO_3, LED_NO_4, // 1행
63      LED_NO_5, LED_NO_6, LED_NO_7, LED_NO_8, // 2행
64      LED_NO_9, LED_NO_10, LED_NO_11, LED_NO_12, // 3행
65      LED_NO_13, LED_NO_14, LED_NO_15, LED_NO_16  // 4행
66    };
67
68    // 키패드 버튼 핀 번호 배열
69    const int buttonPins[16] = {22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32,
70    33, 34, 35, 36, 37}; // PB1 ~ PB16
71    const int buzzerPin = 47; // BUZZER 핀 번호 (A0 사용)
72
73    const int FndSelectTable[6] = {
74      FND1, FND2, FND3, FND4, FND5, FND6
75    };
76
77    const int FndPinTable[8] = {
78      FND_A, FND_B, FND_C, FND_D, FND_E, FND_F, FND_G, FND_DP
79    };
80
81    const int FndNumberTable[10] = {
82      A_BIT | B_BIT | C_BIT | D_BIT | E_BIT | F_BIT, // '0'
83      B_BIT | C_BIT, // '1'
84      A_BIT | B_BIT | D_BIT | E_BIT | G_BIT,  // '2'
85      A_BIT | B_BIT | C_BIT | D_BIT | G_BIT,  // '3'
86      B_BIT | C_BIT | F_BIT | G_BIT, // '4'
87      A_BIT | C_BIT | D_BIT | F_BIT | G_BIT,  // '5'
88      C_BIT | D_BIT | E_BIT | F_BIT | G_BIT,  // '6'
89      A_BIT | B_BIT | C_BIT,  // '7'
90      A_BIT | B_BIT | C_BIT | D_BIT | E_BIT | F_BIT | G_BIT, // '8'
91      A_BIT | B_BIT | C_BIT | F_BIT | G_BIT // '9'
92    };
```

다음으로는 핀 배열 및 초기 설정 부분입니다. LedPinTable에 16개의 LED 핀 번호를 배열로 정의하고 buttonPins의 16개의 KEYPAD 버튼 핀 번호를 배열로 정의합니다. Buzzer의 경우 하나의 핀만 사용하면 되기 때문에 47번 핀을 사용하였습니다. FndSelectTable에서 7-segment display의 자리를 선택하기 위한 핀 번호 배열을 정의하고 FndPinTable에서 7-segment display의 각 segment를 제어하기 위한 핀 번호 배열을 정의합니다. FndNumberTable의 경우 숫자 0~9에 해당하는 7-segment display의 비트 패턴을 정의하였습니다.

```
94    // FND 초기화
95    void FndInit() {
96      int i;
97      for (i = 0; i < MAX_FND; i++) {
98        pinMode(FndSelectTable[i], OUTPUT);     // FND Sel Pin OUTPUT Set
99        pinMode(FndPinTable[i], OUTPUT);        // FND Data Pin OUTPUT Set
100     }
101   }
102
103   // FND 선택
104   void FndSelect(int Position) {
105     int i;
106     for (i = 0; i < MAX_FND_NUM; i++) {
107       if (Position & (1 << i)) {
108         digitalWrite(FndSelectTable[i], LOW);
109       } else {
110         digitalWrite(FndSelectTable[i], HIGH);
111       }
112     }
113   }
114
115   // FND 끄기
116   void FndOff() {
117     int i;
118     for (i = 0; i < MAX_FND; i++) {
119       digitalWrite(FndSelectTable[i], HIGH);
120     }
121     delay(50);
122   }
```

FND의 초기화 함수입니다. FndInit함수는 FND의 핀을 초기화합니다 또한 FndSelectTable의 각 피관 FndPinTable의 각 핀을 OUTPUT모드로 설정합니다. FndSelect함수의 경우 특정 FND를 선택합니다. Position 파라미터에 따라 해당 FND의 선택 핀을 LOW로 설정하고, 나머지 HIGH로 설정합니다. 예를 들어, Position이 1일 때, 첫 번째 FND가 선택됩니다. FndOff함수는 모든 FND를 끄는 동작을 수행하고 FndSelectTable의 각 핀을 HIGH으로 설정하여 모든 FND를 비활성화합니다.

```
125   void FndData(char text) {
126     int i;
127     if (text >= '0' && text <= '9') {
128       for (i = 0; i < 8; i++) {
129         if (FndNumberTable[text - '0'] & (1 << i)) {
130           digitalWrite(FndPinTable[i], HIGH);
131         } else {
132           digitalWrite(FndPinTable[i], LOW);
133         }
134       }
135     } else {
136       // FND에 알파벳 출력
137       byte data;
138       switch (text) {
139         case 'A': data = A_BIT | B_BIT | C_BIT | E_BIT | F_BIT | G_BIT; break;
140         case 'C': data = A_BIT | D_BIT | E_BIT | F_BIT; break;
141         case 'E': data = A_BIT | D_BIT | E_BIT | F_BIT | G_BIT; break;
142         case 'G': data = A_BIT | C_BIT | D_BIT | E_BIT | F_BIT; break;
143         case 'O': data = A_BIT | B_BIT | C_BIT | D_BIT | E_BIT | F_BIT; break;
144         case 'L': data = F_BIT | E_BIT | D_BIT; break;
145         case 'I': data = F_BIT | E_BIT; break;
146         case 'D': data = B_BIT | G_BIT | E_BIT | D_BIT | C_BIT; break;
147         case 'F': data = A_BIT | F_BIT | G_BIT | E_BIT; break;
148         default: data = 0;
149       }
150       for (i = 0; i < 8; i++) {
151         if (data & (1 << i)) {
152           digitalWrite(FndPinTable[i], HIGH);
153         } else {
154           digitalWrite(FndPinTable[i], LOW);
155         }
156       }
157     }
158     delay(1);
```

FndData함수는 FND에 숫자나 문자를 출력하는데 text가 숫자일 경우 FndNumberTable을 참조하여 해당 숫자를 출력하고, 문자인 경우에 switch문을 사용하여 알파벳을 FND에 출력하도록 하였습니다.

```
162 void DrawTextFndGood(const char* text) {
163   int len = strlen(text);  // 문자열의 길이를 가져옴
164   int displayTime = 1000;  // 텍스트를 표시할 총 시간 (밀리초)
165   int refreshInterval = 5; // 텍스트 갱신 간격 (밀리초)
166   int iterations = displayTime / refreshInterval;
167   int buzzerTones[5] = {329, 440, 554, 659, 880}; // 정답일 때 주파수
168
169   for (int j = 0; j < iterations; j++) {
170     for (int i = 0; i < len && i < MAX_FND_NUM; i++) {
171       FndSelect(1 << (MAX_FND_NUM - len + i));  // 오른쪽에서 왼쪽으로 선택
172       FndData(text[len - 1 - i]);      // 문자열의 각 문자를 반대로 출력
173     }
174     tone(buzzerPin, buzzerTones[(j / (iterations / 5)) % 5], 200); // 동시
175     delay(refreshInterval); // 갱신 간격 만큼 대기
176   }
177   FndOff();
178 }
```

DrawTextFndGood함수는 결과적으로는 GOOD이라는 텍스트를 FND에 표시하고 정해진 주파수에 맞는 소리를 Buzzer에서 출력하도록 합니다. 조금 더 디테일하게 보면 문자열 text의 길이를 가져오고, 텍스트를 표시할 총 시간을 설정하고, 갱신 간격을 설정합니다. displayTime을 refreshInterval로 나눠 몇 번 갱신할지를 계산합니다. 루프를 돌며 각 문자에 대해 FND를 선택하고 데이터를 출력합니다. 이와 동시에 정의되어 있는 소리를 출력하고, 갱신 간격 만큼 대기합니다. 모든 작업이 끝나면 FndOff함수를 호출해 FND를 끕니다.

```
void DrawTextFndFail(const char* text) {
  int len = strlen(text);  // 문자열의 길이를 가져옴
  int displayTime = 1000;  // 텍스트를 표시할 총 시간 (밀리초)
  int refreshInterval = 5; // 텍스트 갱신 간격 (밀리초)
  int iterations = displayTime / refreshInterval;
  int failTone = 880; // 오답일 때 주파수

  for (int j = 0; j < iterations; j++) {
    for (int i = 0; i < len && i < MAX_FND_NUM; i++) {
      FndSelect(1 << (MAX_FND_NUM - len + i));  // 오른쪽에서 왼쪽으로 선택
      FndData(text[len - 1 - i]);     // 문자열의 각 문자를 반대로 출력
    }
    tone(buzzerPin, failTone, 200); // 동시에 소리 출력
    delay(refreshInterval); // 갱신 간격 만큼 대기
  }
  FndOff();
}
```

DrawTextFndFail함수의 경우 DrawTextFndGood함수와 거의 비슷한 동작을 수행하긴 하지만 DrawTextFail함수의 경우 FND에 TEXT를 출력할 때 “FAIL”을 출력하고 Good과는 다른 주파수의 소리를 출력합니다.

```
void DrawNumberFnd(int num) {
  FndSelect(1 << 0);  // 첫 번째 위치 선택
  FndData('0' + num);  // 숫자를 문자로 변환하여 출력
}

// 피드백을 제공하는 함수
void provideFeedback(bool correct) {
  if (correct) {
    DrawTextFndGood("GOOD");
    for (int i = 0; i < 16; i++) {
      digitalWrite(LedPinTable[i], HIGH); // 모든 LED 켜기
      delay(100);
      digitalWrite(LedPinTable[i], LOW);  // 모든 LED 끄기
    }
  } else {
    DrawTextFndFail("FAIL");
    for (int i = 0; i < 16; i++) {
      digitalWrite(LedPinTable[i], HIGH); // 모든 LED 켜기
      delay(50);
      digitalWrite(LedPinTable[i], LOW);  // 모든 LED 끄기
    }
  }
}
```

DrawNumberFnd함수의 경우 FndSelect 함수를 호출해 첫 번째 FND를 선택합니다. 이후 숫자 num을 변환하여 FndData함수로 출력하여 숫자를 FND에 출력합니다.ProvideFeedback함수의 경우 사용자의 입력에 따른 정답과 오답 여부에 따라 피드백을 제공합니다. correct가 True일 경우 DrawTExtFndGood함수를 호출해 “GOOD”을 출력하고, 모든 LED를 켰다가 끕니다. 두 번째로 correct가 false일 경우 즉 오답일 경우에는 DrawTextFndFail함수를 호출하여 “FAIL”을 FND에 출력하고 LED를 켰다가 끄는데 이 때, LED가 켜지고 꺼지는 간격을 다르게 설정하였습니다.

```
225   const int ledToButtonMap[16] = {
226     3, 2, 1, 0, 7, 6, 5, 4, 11, 10, 9, 8, 15, 14, 13, 12
227   };
228
229   // 키패드와 LED 매칭 정의 (문서화된 매핑)
230   const int buttonToLedMap[16] = {
231     3, 2, 1, 0, 7, 6, 5, 4, 11, 10, 9, 8, 15, 14, 13, 12
232   };
233
234   // 스테이지 1 패턴 정의
235   const int stage1Pattern1[] = {3, 2, 1, 0, 4, 8, 10, 11}; // LED4, LED3, LED2, LI
236   const int stage1Pattern2[] = {1, 2, 3, 4, 8, 12}; // 다른 패턴1
237   const int stage1Pattern3[] = {0, 1, 2, 3, 5, 9}; // 다른 패턴2
238
239   const int* stage1Patterns[] = {stage1Pattern1, stage1Pattern2, stage1Pattern3};
240   const int stage1Lengths[] = {8, 6, 6}; // 각 패턴의 길이
241
242   // 스테이지 2 패턴 정의
243   const int stage2Pattern1[] = {3, 2, 1, 0, 4, 8, 12, 13, 14, 15}; // LED4, LED3,
244   const int stage2Pattern2[] = {2, 3, 4, 5, 9, 13, 14, 15, 11, 7}; // 다른 패턴1
245   const int stage2Pattern3[] = {1, 0, 4, 8, 12, 13, 14, 15, 7, 3}; // 다른 패턴2
246
247   const int* stage2Patterns[] = {stage2Pattern1, stage2Pattern2, stage2Pattern3};
248   const int stage2Lengths[] = {10, 10, 10}; // 각 패턴의 길이
249
250   // 스테이지 3 패턴 정의
251   const int stage3Pattern1[] = {3, 2, 1, 0, 5, 10, 15}; // LED4, LED3, LED2, LED1,
252   const int stage3Pattern2[] = {2, 1, 0, 3, 6, 11, 10}; // 다른 패턴1
253   const int stage3Pattern3[] = {1, 0, 3, 2, 7, 8, 9, 15, 14}; // 다른 패턴2 st3 st
254
255   const int* stage3Patterns[] = {stage3Pattern1, stage3Pattern2, stage3Pattern3};
256   const int stage3Lengths[] = {7, 7, 9}; // 각 패턴의 길이
```

LedToButtonMap과 buttonToLedMap 배열은 LED와 KEYPAD 간의 매핑을 정의하였습니다. 이 매핑을 통해서 키패드 버튼과 LED를 서로 대응시킬 수 있었습니다. 그래야지 LED에서 출력하는 위치와 KEYPAD의 위치를 동일하게 설정할 수 있기 때문입니다. 다음으로는 각 각 스테이지의 패턴 정의부분입니다. 해당 스테이지가 1스테이지부터 15스테이지까지 있는데 이미지로 첨부하기에 너무 길기 때문에 1스테이지부터 3스테이지까지의 이미지만 첨부하였습니다. stagePattern의 경우 해당 스테이지의 LED패턴을 정의합니다. 즉 출력하여 사용자가 시각적으로 확인할 수 있도록합니다. 그리고 stageButtons의 경우 사용자가 시각적으로 확인한 LED에 맞게 입력 받아야할 입력을 정의합니다. 또한 LED의 패턴은 한 스테이지당 3개의 패턴을 정의하여 하나의 스테이지에서 하나의 패턴만 출력하는 것이 아니라 3개 중의 하나의 패턴을 출력하도록 하였습니다.stageLengths는 해당 패턴의 길이를 의미합니다.

```
294     const int maxSequenceLength = 16; // 최대 순서 길이
295     int userInput[maxSequenceLength]; // 사용자 입력 배열
296     bool ledStatus[16] = {false}; // LED 상태 배열
297
298     bool gameStarted = false;
299     int currentStage = 0;
300
301     //Text LCD 정의
302     const int rs = A0;
303     const int rw = A1;
304     const int e =  A2;
305
306     const int d0 = A8;
307     const int d1 = A9;
308     const int d2 = A10;
309     const int d3 = A11;
310     const int d4 = A12;
311     const int d5 = A13;
312     const int d6 = A14;
313     const int d7 = A15;
```

MaxSequenceLength는 최대 순서 길이를 정의하고, userInput 배열은 사용자의 입력을 저장합니다. ledStatus배열은 각 LED의 상태를 저장합니다. 그 외의 gameStarted, currentStage변수는 게임 상태를 관리합니다. 다음은 LCD에서 TEXT를 출력하기 위한 핀을 정의하였습니다.

```
315    // LCD 초기화 명령어 및 설정
316    void lcdCommand(uint8_t command) {
317      digitalWrite(rs, LOW);
318      digitalWrite(rw, LOW);
319      digitalWrite(e, HIGH);
320
321      digitalWrite(d0, command & 0x01);
322      digitalWrite(d1, command & 0x02);
323      digitalWrite(d2, command & 0x04);
324      digitalWrite(d3, command & 0x08);
325      digitalWrite(d4, command & 0x10);
326      digitalWrite(d5, command & 0x20);
327      digitalWrite(d6, command & 0x40);
328      digitalWrite(d7, command & 0x80);
329
330      delayMicroseconds(1); // Enable pulse must be >450ns
331      digitalWrite(e, LOW);
332      delayMicroseconds(50); // Commands need > 37us to settle
333    }
335    // 문자 데이터 전송 함수
336    void lcdData(uint8_t data) {
337      digitalWrite(rs, HIGH);
338      digitalWrite(rw, LOW);
339      digitalWrite(e, HIGH);
340
341      digitalWrite(d0, data & 0x01);
342      digitalWrite(d1, data & 0x02);
343      digitalWrite(d2, data & 0x04);
344      digitalWrite(d3, data & 0x08);
345      digitalWrite(d4, data & 0x10);
346      digitalWrite(d5, data & 0x20);
347      digitalWrite(d6, data & 0x40);
348      digitalWrite(d7, data & 0x80);
349
350      delayMicroseconds(1); // Enable pulse must be >450ns
351      digitalWrite(e, LOW);
352      delayMicroseconds(50); // Data needs > 37us to settle
353    }
```

LcdCommand함수는 LCD에 명령어를 전송합니다 Command 파라미터에 따라 각 데이터 핀을 설정하고, ‘e’ 핀을 통해서 명령어를 LCD로 전송합니다. LcdData함수 또한 data 파라미터에 따라 각 데이터 핀을 설정하고, ‘e’핀을 설정하고 ‘e’핀을 통해 데이터를 LCD에 전송합니다.

```
// LCD 초기화 함수
void lcdInit() {
  pinMode(rs, OUTPUT);
  pinMode(rw, OUTPUT);
  pinMode(e, OUTPUT);
  pinMode(d0, OUTPUT);
  pinMode(d1, OUTPUT);
  pinMode(d2, OUTPUT);
  pinMode(d3, OUTPUT);
  pinMode(d4, OUTPUT);
  pinMode(d5, OUTPUT);
  pinMode(d6, OUTPUT);
  pinMode(d7, OUTPUT);

  delay(50); // Wait fo
```

```
371    lcdCommand(0x30); // Func
372    delayMicroseconds(4500);
373    lcdCommand(0x30); // Func
374    delayMicroseconds(150); /
375    lcdCommand(0x30); // Func
376
377    lcdCommand(0x38); // Func
378    lcdCommand(0x0C); // Disp
379    lcdCommand(0x06); // Entr
380    lcdCommand(0x01); // Clea
381    delayMicroseconds(2000);
382  }
```

LcdInit함수는 LCD를 초기화합니다. 각 핀을 OUTPUT모드로 설정하고, 여러 명령어를 전송해 LCD의 초기 설정을 완료합니다.

```
445    void lcdPrint(const char* str) {
446      while (*str) {
447        lcdData(*str++);
448      }
449    }
450
451    // LCD에 현재 스테이지를 출력하는 함수
452    void displayStage(int stage) {
453      lcdCommand(0x01); // Clear display
454      delay(2); // Wait for the clear command to execute
455
456      lcdPrint("Stage : ");
457      if (stage < 10) {
458        lcdData('0' + stage); // 한 자리 숫자
459      } else {
460        lcdData('0' + stage / 10); // 첫 번째 자리 숫자
461        lcdData('0' + stage % 10); // 두 번째 자리 숫자
462      }
463    }
```

```
400    // 초기 상태로 돌아가는 함수
401    void resetToInitialState() {
402      // 모든 LED를 켭니다.
403      for (int i = 0; i < 16; i++) {
404        digitalWrite(LedPinTable[i], HIGH);
405      }
406
407      // 게임 시작 대기
408      gameStarted = false;
409      currentStage = 0; // 첫 번째 스테이지로 초기화
410      displayStage(currentStage); // 초기화면에 Stage : 0 출력
411
412      while (!gameStarted) {
413        for (int i = 0; i < 16; i++) {
414          if (digitalRead(buttonPins[i]) == LOW) {
415            gameStarted = true;
416            break;
417          }
418        }
419      }
420
421      // 게임 시작 시 currentStage를 1로 설정
422      currentStage = 1;
```

```
424      // 카운트다운
425      for (int i = 5; i > 0; i--) {
426        if (i == 5) {
427          displayStage(currentStage); // Stage 증가 시점 변경
428        }
429        DrawNumberFnd(i);
430        tone(buzzerPin, 700, 200); // 1초마다 700Hz 버저 울리기
431        delay(1000); // 1초 대기
432      }
433
434      // 게임 시작 시 모든 LED를 끕니다.
435      turnOffAllLeds();
436    }
```

LcdPrint함수는 문자열을 LCD에 출력합니다. 문자열의 각 문자를 lcdData함수를 통해 LCD로 전송합니다. display Stage 함수는 현재 스테이지를 LCD에 출력합니다. 먼저 LCD를 초기화하고, “Stage : 해당스테이지”문자열과 함께 현재 스테이지 번호를 LCD에 출력합니다. 즉 사용자가 진행하고 있는 해당 스테이지에서 사용자가 해당 스테이지에

정의된 패턴에 맞게 입력을 하였을 경우 “GOOD”을 출력하며 다음 스테이지로 넘어가는데 이 때 “Stage : 1” 이었다면 “Stage : 2”를 출력하는 것입니다. resetToInitialState 함수의 경우 초기 상태로 돌아가는 동작을 수행합니다. 모든 LED를 켜고 게임 시작을 대기하며, 버튼이 눌릴 때 까지 기다리다가 게임이 시작되면 currentStage를 1로 설정하고, 카운트 다운을 표시합니다. 카운트 다운이 끝나면 모든 LED를 끄는 동작을 수행합니다.

```
511  void celebrationEffect() {
512    int buzzerTones[] = {262, 294, 330, 349, 392, 440, 494, 523}; // 축하음 주파
513
514    for (int i = 0; i < 3; i++) { // 3초 동안 이펙트 반복
515      for (int j = 0; j < 16; j++) {
516        digitalWrite(LedPinTable[j], HIGH);
517        tone(buzzerPin, buzzerTones[j % 8], 100); // 각 LED에 대해 다른 톤 재생
518        delay(50);
519        digitalWrite(LedPinTable[j], LOW);
520      }
521      for (int j = 15; j >= 0; j--) {
522        digitalWrite(LedPinTable[j], HIGH);
523        tone(buzzerPin, buzzerTones[j % 8], 100); // 각 LED에 대해 다른 톤 재생
524        delay(50);
525        digitalWrite(LedPinTable[j], LOW);
526      }
527    }
528    noTone(buzzerPin); // 모든 소리 끄기
529  }
```

celebrationEffect함수는 15단계를 통과했을 때 정해져있는 여러개의 주파수의 음들을 Buzzer에서 출력합니다. 또한 LED에서는 다양한 모션의 LED를 출력하여 15단계를 통과했음을 사용자에게 알려줍니다.

```
438  // 모든 LED 끄기
439  void turnOffAllLeds() {
440    for (int i = 0; i < 16; i++) {
441      digitalWrite(LedPinTable[i], LOW);
442    }
443  }
444
445  void setup() {
446    // LED 핀과 키패드 버튼 핀을 설정
447    for (int i = 0; i < 16; i++) {
448      pinMode(LedPinTable[i], OUTPUT);
449      pinMode(buttonPins[i], INPUT_PULLUP); // 키패드 버튼 핀을 입력으로 설정
450    }
451    pinMode(buzzerPin, OUTPUT);
452    FndInit(); // FND 초기화
453    randomSeed(analogRead(0)); // 무작위 시드 설정
454
455    lcdInit(); // LCD 초기화
456    resetToInitialState(); // 초기 상태 설정
457  }
```

turnOffAllleds함수는 모든 LED를 끄는 동작을 수행합니다. LEDPintable의 각 핀을 LOW로 설정하여 모든 LED를 끄도록합니다. setup함수의 경우 초기 설정에 대한 동작을 수행합니다. LED핀과 KEYPAD의 BUTTON핀을 설정하고 Buzzer PIN을 OUTPUT모드로 설정합니다. FndInit함수를 호출해 FND를 초기화하고 randomSeed함수를 호출하여 무작위 시드를 설정합니다. LcdInit함수를 호출해 LCD를 초기화하고 resetToInitialState함수를 호출하여 초기상태로 설정합니다.

```
546   int getStageLedDuration(int stage) {
547     if (stage >= 1 && stage <= 5) {
548       return 700; // 1스테이지부터 5스테이지까지는 0.7초 동안 유지
549     } else if (stage >= 6 && stage <= 10) {
550       return 500; // 6스테이지부터 10스테이지까지는 0.5초 동안 유지
551     } else if (stage >= 11 && stage <= 15) {
552       return 300; // 11스테이지부터 15스테이지까지는 0.3초 동안 유지
553     }
554     return 1000; // 기본 유지 시간 1초
555   }
556
557   // LED 패턴을 표시하는 함수
558   void showPattern(const int* pattern, int length, int duration) {
559     // 모든 LED를 끕니다.
560     turnOffAllLeds();
561     // 모든 LED를 한꺼번에 켭니다.
562     for (int i = 0; i < length; i++) {
563       digitalWrite(LedPinTable[pattern[i]], HIGH);
564     }
565     delay(duration); // 지정된 시간 동안 켜진 상태 유지
566     // 모든 LED를 끕니다.
567     turnOffAllLeds();
568   }
```

getStageLedDuration 함수는 각 스테이지의 LED유지 시간을 반환합니다. 스테이지에 따라서 유지 시간이 다르도록 설정하였습니다. 이런 방식을 통해서 사용자가 기억하고 입력하는데 있어서 난이도를 설정하였습니다. 1~5단계는 하 단계로 LED가 0.7초 동안 켜졌다가 꺼지고 6~10단계는 LED가 0.5초 동안 유지됐다가 꺼지고, 11~15단계의 경우 LED가 0.3초 켜졌다가 꺼지도록 설정하였습니다. ShowPattern 함수는 주어진 패턴 즉 정의된 패턴의 LED를 지정된 시간 동안 키는 동작을 수행합니다. 모든 LED를 껐다가 패턴 배열의 각 LED를 키고 지정된 시간 동안 유지하였다가 지정된 시간이 끝나면 모든 LED를 끄는 동작을 수행합니다.

```
571   bool getUserInputWithTimeout(int* userInput, int length, int timeout) {
572     int count = 0;
573     unsigned long startTime = millis();
574     bool buzzerPlayed[3] = {false, false, false}; // 3, 2, 1 초에서 버저가 울렸는지 확인하기 위한 배열
575
576     while (count < length) {
577       unsigned long elapsedTime = (millis() - startTime) / 1000;
578       int remainingTime = timeout - elapsedTime;
579
580       if (remainingTime >= 0) {
581         DrawNumberFnd(remainingTime); // 남은 시간 FND에 표시
582       }
583
584       if (remainingTime == 3 && !buzzerPlayed[0]) {
585         tone(buzzerPin, 700, 200); // 3초에 도달하면 Buzzer 소리
586         buzzerPlayed[0] = true;
587       }
588       if (remainingTime == 2 && !buzzerPlayed[1]) {
589         tone(buzzerPin, 700, 200); // 2초에 도달하면 Buzzer 소리
590         buzzerPlayed[1] = true;
591       }
592       if (remainingTime == 1 && !buzzerPlayed[2]) {
593         tone(buzzerPin, 700, 200); // 1초에 도달하면 Buzzer 소리
594         buzzerPlayed[2] = true;
595       }
596
597       if (remainingTime <= 0) { // 제한 시간 초과 확인
598         noTone(buzzerPin); // Buzzer 끄기
599         return false;
```

```
602       for (int i = 0; i < 16; i++) {
603         if (digitalRead(buttonPins[i]) == LOW) {
604           userInput[count] = i;
605           digitalWrite(LedPinTable[buttonToLedMap[i]], HIGH); // 키패드에 해당하는 LED 켜기
606           ledStatus[buttonToLedMap[i]] = true;
607           count++;
608           delay(300); // 디바운싱을 위한 지연
609           while (digitalRead(buttonPins[i]) == LOW); // 버튼이 떼어질 때까지 대기
610         }
611       }
612     }
613     noTone(buzzerPin); // Buzzer 끄기
614     return true;
615   }
```

getUserInputwithTimeout함수의 경우 사용자의 입력을 제한 시간 내에 받도록 합니다. startTime을 기록하고 사용자가 입력한 KEYPAD BUTTON을 userInput 배열에 저장합니다. KEYPAD를 눌렀을 때 해당 KEYPAD와 매칭되는 LED를 켜고, ledStatus 배열에 기록합니다. 제한 시간이 초과되면 False를 반환합니다.또한 각 스테이지에 정의되어 있는 카운트다운 제한시간에서 3초에 도달했을 때 3초남았다는 경고음을 출력합니다. 그래서 해당 카운트에 맞게 3초부터 시작하여 제한시간이 끝날때까지 Buzzer에서 소리를 출력합니다.

```
514   // 사용자의 입력이 정답인지 확인하는 함수
515   bool checkUserInput(const int* pattern, const int* userInput, int length) {
516     for (int i = 0; i < length; i++) {
517       if (!ledStatus[pattern[i]]) {
518         return false;
519       }
520     }
521     return true;
522   }
```

.CheckUserInput 함수는 사용자의 입력이 정답인지를 확인합니다. 정의되어 있는 Pattern 배열과 userInput배열을 비교하여 일치하지 않는 항목이 있다면 False를 반환하도록 설정하였습니다.

```
628 void getStageInfo(int stage, const int* &pattern, int &length) {
629     int patternIndex = random(0, 3); // 랜덤한 패턴 선택
630     switch (stage) {
631       case 1:
632         pattern = stage1Patterns[patternIndex];
633         length = stage1Lengths[patternIndex];
634         break;
635       case 2:
636         pattern = stage2Patterns[patternIndex];
637         length = stage2Lengths[patternIndex];
638         break;
639       case 3:
640         pattern = stage3Patterns[patternIndex];
641         length = stage3Lengths[patternIndex];
642         break;
643       case 4:
644         pattern = stage4Patterns[patternIndex];
645         length = stage4Lengths[patternIndex];
646         break;
647       case 5:
648         pattern = stage5Patterns[patternIndex];
649         length = stage5Lengths[patternIndex];
650         break;
```

```
610   // 각 스테이지의 시간 제한을 반환하는 함수
611   int getStageTimeLimit(int stage) {
612     if (stage >= 1 && stage <= 5) {
613       return 9; // 1스테이지부터 5스테이지까지는 시간 제한 9초
614     } else if (stage >= 6 && stage <= 10) {
615       return 7; // 6스테이지부터 10스테이지까지는 시간 제한 7초
616     } else if (stage >= 11 && stage <= 15) {
617       return 5; // 11스테이지부터 15스테이지까지는 시간 제한 5초
618     }
619     return 9; // 기본 시간 제한 9초
620   }
```

패턴과 패턴의 길이를 반환합니다. `stage` 파라미터에 따라 해당 스테이지의 패턴과 길이를 설정합니다. `random(0,3)`은 하나의 스테이지에 패턴이 3개 정의되어 있는데, 항상 똑같은 LED 패턴만 출력하는 것이 아니라, 3개의 패턴 중에서 랜덤으로 하나의 패턴을 선택하여 출력을 수행합니다. 선택된 패턴은 `pattern`과 `length` 변수에 저장됩니다.`getStageTimeLimit` 함수는 각 스테이지의 시간 제한을 반환합니다. 스테이지에 따라 시간 제한을 다르게 주어서 사용자가 난이도가 올라갈수록 어려워진다는 것을 알 수 있도록 합니다. 예를 들어, 1에서 5단계까지는 9초, 6에서 10단계까지는 7초, 11에서 15단계까지는 5초의 시간 제한이 있습니다. 종합해보면 1에서 5단계까지는 LED는 1초 동안 출력하였다가 꺼지고 9초 이내로 사용자가 LED에 해당하는 입력을 수행해야합니다. 6에서 10단계는 LED를 0.8초 동안 출력하였다가 꺼지고 7초 이내로 사용자가 LED에 해당하는 입력을 수행해야합니다. 11에서 15단계는 LED를 0.5초 동안 출력하였다가 꺼지고 5초 이내로 사용자가 LED에 해당하는 입력을 수행해야합니다.

```
711   void playStage(int stage) {
712     const int* pattern;
713     int length;
714
715     getStageInfo(stage, pattern, length);
716     if (pattern == nullptr || length == 0) {
717       return; // 잘못된 스테이지일 경우 함수 종료
718     }
719
720     int timeLimit = getStageTimeLimit(stage); // 스테이지에 따른 시간
721     int ledDuration = getStageLedDuration(stage); // 스테이지에 따른
722
723     memset(ledStatus, false, sizeof(ledStatus)); // LED 상태 초기화
724
725     showPattern(pattern, length, ledDuration);
726
727     if (getUserInputWithTimeout(userInput, length, timeLimit)) { //
728       if (checkUserInput(pattern, userInput, length)) {
729         provideFeedback(true); // 정답일 때 긍정적 피드백
730         delay(1000);  // 1초 동안 LED 켜진 상태 유지
731         turnOffAllLeds();  // LED 끄기
732
733         if (stage == 15) { // 15스테이지를 클리어했을 경우
734           lcdCommand(0x01); // Clear display
735           delay(2); // Wait for the clear command to execute
736           lcdPrint("You've all cleared");
737           lcdCommand(0xC0); // Move cursor to the second line
738           lcdPrint("Congratulations");
739           celebrationEffect(); // 축하 LED 이펙트
740           delay(3000); // 3초 대기
741           resetToInitialState(); // 초기 상태로 돌아가기
742           return; // 첫 번째 스테이지로 돌아가기 위해 함수 종료
743         }
```

```
746             for (int i = 5; i > 0; i--) {
747               if (i == 5) {
748                 currentStage++; // Stage 증가 시점 변경
749                 displayStage(currentStage);
750               }
751               DrawNumberFnd(i);
752               tone(buzzerPin, 700, 200); // 1초마다 700Hz 버저 울리기
753               delay(1000); // 1초 대기
754             }
755           } else {
756             provideFeedback(false); // 틀렸을 때 부정적 피드백
757             delay(1000);
758             resetToInitialState(); // 초기 상태로 돌아가기
759             return; // 첫 번째 스테이지로 돌아가기 위해 함수 종료
760           }
761         } else {
762           provideFeedback(false); // 시간 초과 시 부정적 피드백
763           delay(1000);
764           resetToInitialState(); // 초기 상태로 돌아가기
765           return; // 첫 번째 스테이지로 돌아가기 위해 함수 종료
766         }
767       }
768
769       void loop() {
770         playStage(currentStage);
771       }
```

playstage함수는 현재 스테이지의 루프를 관리합니다. 순서는 다음과 같이 진행됩니다. getStageINfo함수를 호출하여 스테이지의 패턴과 길이를 가져오고 StageTimeLimit함수로 시간 제한을 설정합니다 다음으로 getStageLedDuration 함수로 LED유지 시간을 설정합니다 이후에 showPattern으로 패턴을 LED에 표시하고 getUserInputwithTimeout을 통해 사용자로부터의 KEYPAD입력을 받습니다. CheckUserInput함수로부터 사용자의 입력이 출력된 LED와 매칭되는지 확인합니다. 정답일 경우에 ProvideFeedback으로 Good을 출력하고 다음 스테이지로 넘어가고 오답일 경우에 FAIL을 출력하고 게임 시작 전인 초기화면으로 돌아갑니다. displayStage로 스테이지를 업데이트합니다.

●총평

종합적으로 해당 프로젝트는 LED패턴을 정의하고 정의되어 있는 LED의 위치와 KEYPAD의 위치를 동일하게 매핑 하여 사용자가 KEYPAD를 눌렀을 때 해당 KEYPAD에 매칭되어 있는 LED가 켜지도록 하여 정의된 패턴의 LED가 출력되면 사용자가 출력된 LED를 기억하였다가 KEYPAD를 통해 입력하여 맞추는 동작을 수행합니다. 각 스테이지별로 난이도가 설정되어 있어서 스테이지가 올라가면 올라갈수록 사용자는 더 높은 집중력과 반응속도 그리고 기억력을 요구하여 능력 향상과 더불어 재미를 느낄 수 있도록 하였습니다.

●프로젝트 구현 시 문제점 & 문제 극복과 코드개선

현재 코드와 달리 이전의 코드를 작성하였을 때는 모든 스테이지에 대한 로직을 loop함수 내에서 직접 처리하여, 각 스테이지마다 반복되는 코드가 많았습니다. 이렇게 하다보니 코드의 길이도 1000줄이 넘게 증가하였고 메모리 또한 많이 사용하였습니다. 무엇보다 모든 스테이지에 대한 동작을 loop에서 실행하다보니 각 스테이지 사이에 난이도를 조절하기 위한 동작들 구현 또한 제대로 이루어지지 않고 서로 다른 동작에 영향을 미치는 치명적인 오류가 발생하였고 유지보수 또한 굉장히 어려웠습니다. 하지만 조원들과 함께 머리를 맞대어 해결방안을 찾았고, 각 함수를 분할하여 기능을 독립적으로 처리하고 필요할 때 호출하여 사용하도록 수정하였습니다. 이러한 수정을 통해 코드 또한 500줄 가까이 줄었으며, 메모리 사용량 또한 대폭 감소하였고, 반복되는 코드가 줄어들어 새로운 스테이지를 추가하거나 기존의 스테이지를 수정할 때 유지보수가 용이해지고 가독성이 향상되었습니다. 또한 각 함수가 독립적으로 동작하여 다른 프로젝트에서도 사용할 수 있으므로 재사용성 또한 증가하였고 버그 수정과 기능 추가 시 다른 코드의 동작에

영향을 미치 않도록하여 모듈화 그리고 확장성 또한 증가하였습니다.

●프로젝트 수행 과정에서 느낀 점

이번 마이크로프로세서 프로젝트를 통해서 혼자하는 것과 조원과 같이 머리를 맞대는 것에 대해 정말 큰 차이를 느꼈으며, 하드웨어와 소프트웨어가 결합된 프로젝트를 통해 실용적인 학습 경험을 쌓을 수 있었습니다. LED, FND, KEYPAD, BUZZER, LCD 등 다양한 컴포넌트들을 다루면서 실제 응용 프로그램을 만드는 능력을 배울 수 있었습니다.

이상으로 4조 FINAL PROJECT 보고서 마무리하도록 하겠습니다.

감사합니다.