PROJECT REPORT

<WAV Player , LineTracer>

교과목	임베디드 시스템 설계
프로젝트 기간	2021.09.01 ~ 2021.12.31
이름	신건희

INDEX

1. WAV Player

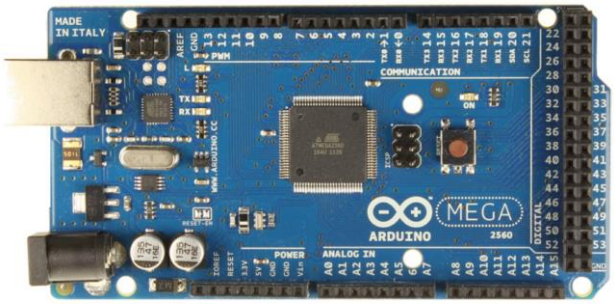
1. Abstract

전공 과목인 임베디드 시스템 설계를 수강하며 진행한 WAV Player 프로젝트이다. 16bit/8bit, 22050Hz/44100Hz, Stereo/Mono 조합으로 추출된 8개의 WAV 파일을 SD 카드를 통해 입력 받고 Atemga 2560, Amp Board, Speaker를 통해 재생한다. Timer/Counter, PWM, Interrupt 등 다양한 기능이 사용된다. 프로젝트 제출시에는 8bit WAV 파일이 느리게 재생되고 16bit WAV파일을 재생되지 않았다. 하지만 종강 후 Peripheral Board와 Amp Board를 다시 대여하여 많은 조사와 고민 끝에 모든 조합의 WAV 파일을 재생할 수 있었다.

2. Hardware

1. Atmega Arduino 2560

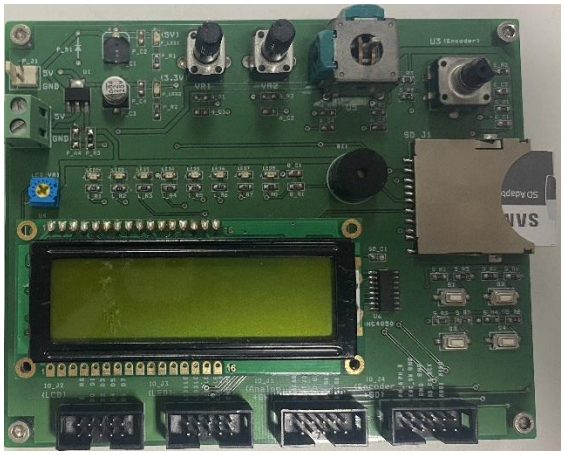
- 54개의 digital input/output 핀이 있으며 그 중 15개는 PWM 출력을 갖고 있으며 16개의 ADC 입력을 갖고 있다.
- Clock Speed는 16MHz이며 Flash Memory는 256KB이다.



<FIG.1 Atmeag Arduino 2560 >

2. Peripheral Board

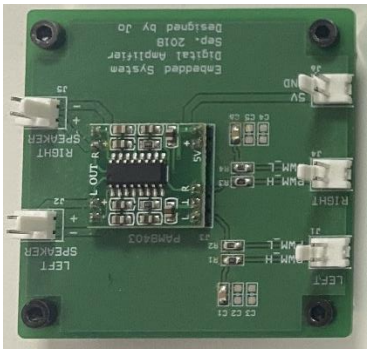
- 다양한 부품을 모아둔 Board
- Board에는 다음의 부품이 있다.
  - 2 Potentiometers, Joystick, Rotary encoder with a push button, 8 LEDS, 4 switches, 16X2 character LCD, SD card connector, Buzzer
  - 본 프로젝트에서는 SD card connector, Rotary encoder를 사용한다.
- 인하대학교 전기공학과 이영삼 교수 개발



<FIG.2 Peripheral Board>

3. Amp board

- Atmega 2560에는 DAC 출력 기능이 없다. 그러므로 WAV 음악 정보를 아날로그 형식으로 출력하는 것은 불가능하다.
- 그래서 8 or 16bit 음악 정보를 PWM 형태로 변환하여 Amp Board에 전달한다.
- Amp Board는 PWM 신호에 가중치 합산과 LPF를 사용하여 아날로그에 비슷한 신호를 생성하여 스피커에 전달한다.



<FIG.3 Amp Board>

3. Arduino Software

1. Timer/Counter

- Atmega 2560에는 2개의 8bit 그리고 4개의 16bit Timer/Counter pin이 있다.
- 본 프로젝트에서는 8bit Timer/Counter를 사용한다. WAV 음악 정보를 PWM 형태로 보내기 위해 해당 레지스터를 조작하여 Fast-PWM Mode, Clear on Compare Match, No prescale factor로 설정한다. PWM Frequency는 $\frac{16\times10^6}{(1+(2^8-1)}=62.5KHz$가 된다.

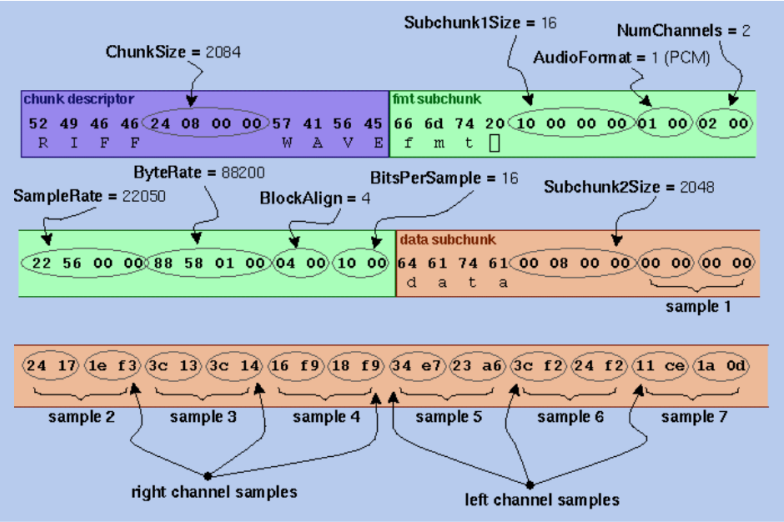
- 16bit Timer/Counter로 PWM을 만들면 안 되는 이유는 16bit Timer/Counter로 PWM 신호를 보내게 되면 Frequency가 $\frac{16\times10^6}{(1+(2^{16}-1)} = 244Hz$가 되어 22.05KHz, 44.1KHz WAV 파일을 재생하지 못한다.

2. Interrupt

- 본 프로젝트에서는 Timer/Counter1 Compare Match A와 2개의 External interrupt를 사용한다.
- Timer/Counter1 Compare Match A에서는 WAV 음악 정보를 바탕으로 OCR 값을 조작하여 PWM Duty ratio를 결정한다.
- Peripheral Board의 Rotary encoder에는 누르는 버튼과 좌우로 회전할 수 있다. Peripheral Board에서 앞의 두가지 신호가 발생 했을 때 External interrupt를 발생하도록 한다. External Interruption Routine에서는 WAV 음악 파일 재생/멈춤 기능과 파일 변경 기능이 있다.

3. WAV 파일 읽기

- 본 프로젝트에서는 Stereo/Mono, 8/16bit, 22.05/44.1KHz가 조합된 8가지 종류의 WAV 음악 파일을 재생한다.
- 그러므로 WAV 음악 파일을 재생하기 위해서는 가장 먼저 WAV파일의 형태를 파악해야 하며 그 정보는 다음 3개의 정보를 통해 확인할 수 있다.
  - NumChannels(22th Byte~23th Byte)
    해당 16bit 정보를 통해 WAV 음악 파일이 Stereo or Mono 형태로 구성되어 있다는 것을 확인할 수 있다. Fast PWM Mode로 설정되어 있는 Timer/Counter 2,4를 이용하여 PWM 신호를 만드는데 Stereo일 경우 OCR 값이 모두 다르게 되고 Mono일 경우 OCR2A=OCR4A and OCR2B=OCR4B가 된다.
  - SampleRate(24th byte~27th byte)
    해당 32bit 정보를 통해 WAV 음악 파일의 Frequency이 22.05Hz or 44.1Hz 형태로 구성되어 있다는 것을 확인할 수 있다. Timer/Counter1 Compare Match A Interruption 에서 PWM duty ration를 바꿔주는데 WAV 음악 파일 Frequency에 따라 OCR1A 값을 바꾸어 Interruption Frequency를 설정한다.
  - BitPerSample(34th byte~35th byte)
    해당 16bit 정보를 통해 WAV 음악 파일이 8bit or 16bit로 구성되어 있다는 것을 확인할 수 있다. 8bit 일 경우 음악 정보를 PWM(L)에 그대로 설정하고 PWM(H)는 0으로 한다. 반면에 16bit 일 경우 signed 정보를 unsigned로 변환한 뒤 PWM(H), PWM(L)에 알맞게 설정한다.



<FIG.4 Example: First 72 Bytes of WAV File>

4. SD카드에서 WAV 파일 정보 가져오기

- WAV 음악 파일이 들어있는 SD카드를 Peripheral Board 에 삽입하고 Atemga 2560을 통해 정보를 가져온다.
- Arduino에서 제공하는 SD Library를 사용했다. 사용한 Class는 다음과 같다
  - SD.open('filename'): SD 카드 내에서 해당 디렉토리에 있는 파일에 접근한다. 본 프로젝트에서는 여러 음악 파일을 접근하므로 다른 파일에 접근하기 전에 반드시 File.close()를 통해 이전 파일을 닫아줘야 한다.
  - File.read(buf,len): Open한 파일에서 len × $byte$ 만큼의 정보를 buf에 저장하고 다음 byte위치르 return한다.
  - File.opennextfile(): 현대 Open된 파일에서 다음 파일로 넘어가는 Class
  - File.name(): File의 제목을 string 형태로 받아온다.

4. Algorithm

1. void printFilename(File dir)

- SD 카드에 들어있는 WAV 파일의 이름을 Music_name 배열에 저장하는 함수이다.
- SD 카드에 있는 파일 중 "WAV"로 끝나는 파일을 찾은 뒤 string 형태로 이름을 저장한다.

- LCD, Serial monitor에 표시하기 위해 char 형태로 변한 뒤 저장한다.

==========

```
void printDirectory(File dir) {   // stores the file name of the music
  String fname;
  while (true) {
    File entry =  dir.openNextFile(); // open the next file in SD card
    if (! entry) {                    // break if there's no next file
      break;
    }
    if (entry.isDirectory()) {
    }
    else {
      fname = entry.name();
      if (fname.endsWith("WAV")) {    // select only wav file in SD card
        fname.toCharArray(Music_name[music_num], fname.length()+1); // convert
string to char arrangement to display on LCD
        music_num++;
      }
    }
  }
}
```

==========

2. void music_info(File myfile)

- WAV flie의 헤더 부분에 파일에 대한 정보가 들어있다.
- NumChannels(22th Byte~23th Byte)
  16bit data로 구성되어 있으며 little endian이므로 두번째 byte를 왼쪽으로 8bit shift 한 뒤 bitwise OR 연산을 한다. 값이 1이면 mono이며 2이면 stereo이다.
- SampleRate(24th byte~27th byte)
  32bit data로 구성되어 있으며 little endian이므로 왼쪽으로 bit shift와 bitwise OR 연산을 반복하여 32bit 데이터로 변환한다. 본 프로젝트에서 재생하는 WAV 파일은 22.05Hz or 44.1KHz 이므로 가중치가 적은 곳부터 16bit만 고려해도 무방하므로 0x0000FFFF와 bitwise AND 연산을 통해 오류의 가능성을 줄였다.
- BitPerSample(34th byte~35th byte)
  16bit data로 구성되어 있으며 little endian이므로 두번째 byte를 왼쪽으로 8bit shift 한 뒤 bitwise OR 연산을 한다. 값이 8이면 8-bit이며 16이면 16-bit이다.
- myfile.read()는 WAV file에서 1byte의 정보를 읽고 다음 byte 위치를 return한다.

==========

```
void music_info(File myfile)      // reads informations of music from header of
WAV file
{
```

```
    i=0;
    while (1) {

    if (i >= 22 && i <= 23) {
      NumChannels[i - 22] = myfile.read();
    }

    else if (i > 23 && i < 28) {
      SampleRate[i - 24] = myfile.read();
    }

    else if (i >= 34 && i <= 35) {
      BitPerSample[i - 34] = myfile.read();
    }
    else if (i > 39 && i < 44) {
      Subchunk2Size[i - 40] = myfile.read();
    }
    else {
      myfile.read();
    }
    i++;
    if (i == 44) {
      break;
    }
    delay(50);
  }

//-----------save BitPerSample data to variables bps--------//
  uint32_t data1 = 0x00000000;
  data1 |= (BitPerSample[0] | BitPerSample[1] << 8); //the data form is little
endian
  Serial.print("Bit per Sampling :");
  Serial.println(data1);
  bps = data1;
//-------------------------------------------------------------//

//------------save NumChannels data to variables nc---------//
  uint32_t data2 = 0x00000000;
  data2 |= (NumChannels[0] | NumChannels[1] << 8); //the data form is little
endian
  Serial.print("Num_Chan :");
  Serial.println(data2);
  nc = data2;
//-------------------------------------------------------------//

  data1 = 0x00000000;
  data2 = 0x00000000;

//----------save SampleRate data to variables sr-----------//
  data1 = (SampleRate[0] | SampleRate[1] << 8);
  data2 |= (SampleRate[2] | SampleRate[3] << 8);
  data1 = data2 << 16 | data1;  //the data form is little endian
  data1 = 0x0000FFFF & data1;   // Correct incorrect values that may occur due
to bitwise AND operation.
  Serial.print("Sampling_Rate :");
  Serial.println(data1);
```

```
  sr = data1;
//-------------------------------------------------------//


//----------display Subchunk2Size data on the serial monitor-----------//
  data1 = 0x00000000;
  data2 = 0x00000000;
  data1 |= (Subchunk2Size[0] | Subchunk2Size[1] << 8);
  data2 |= (Subchunk2Size[2] | Subchunk2Size[3] << 8);
  data1 = data2 << 16 | data1; // the data form is little endian
  data1 = 0x0000FFFF & data1; // Correct incorrect values that may occur due to
bitwise AND operation.
  Serial.print("total music data :");
  Serial.println(data1);
//-----------------------------------------------------------------//

}
```

==========

3. void ExtInterruptInit()

- Peripheral Board의 Rotary Encoder에서 신호가 발생하면 External interruption이 수행되도록 설정하는 함수이다.
- Atmega 2560 INT1 pin에는 Peripheral Board의 PH_A와 연결되어 있으며 PH_A는 Rotary Encoder switch를 눌렀을 때 발생한다.
- Atmega 2560 INT3 pin에는 Peripheral Board의 PH_ㅠ와 연결되어 있으며 PH_ㅠ는 Rotary Encoder를 돌렸을 때 발생한다. 오른쪽으로 돌리면 신호 0이 전달되며 왼쪽으로 돌렸을 때는 1이 전달된다.

==========

```
void ExtInterruptInit(){          // set up ExtInteruupt

//------ INT1 Interrupt / falling edge detection ------//
  EICRA &= ~_BV(ISC10);
  EICRA |= _BV(ISC11);  // falling edge
  EIFR |=_BV(INTF1);    // Clear INT1 Flag
  EIMSK |= _BV(INT1);   // Enable INT1
//-------------------------------------------------------//

//------ INT3 Interrupt / falling edge detection ------//
  EICRA &= ~_BV(ISC30);
  EICRA |= _BV(ISC31);  // falling edge
  EIFR |=_BV(INTF3);    // Clear INT3 Flag
  EIMSK |= _BV(INT3);   // Enable INT3
//-------------------------------------------------------//
}
```

==========

4. void Timer_Counter2(), void Timer_Counter4()

- 2개의 8bit Timer/Counter를 이용하여 PWM을 발생한다.
- 재생하려는 WAV 음악 파일의 frequency인 22.05KHz, 44.1KHz 보다 높게 설정하기 위해 16bit가 아닌 8bit를 사용하고 no prescale factor로 설정했다.

==========

```
void Timer_Counter2() {           // set up 8-bit Timer/Counter2

  TCCR2B &= ~_BV(WGM22);
  TCCR2A |= _BV(WGM21);   // WGM22=0, WGM21=1, WGM20=1
  TCCR2A |= _BV(WGM20);   // Fast PWM, 8-bit / TOP : 0x00FF

  TCCR2A |= _BV(COM2A1);  // COM2A1=1, COM2A0=0
  TCCR2A &= ~_BV(COM2A0); // clear OC2A on Compare Match, set OC2A at TOP

  TCCR2A |= _BV(COM2B1);  // COM2B1=1, COM2B0=0
  TCCR2A &= ~_BV(COM2B0); // clear OC2B on Compare Match, set OC2B at TOP

  TCCR2B &= ~_BV(CS22);
  TCCR2B &= ~_BV(CS21);   // CS22=0,CS21=0,CS20=1
  TCCR2B |= _BV(CS20);    // No prescale factor

  TCNT2 = 0x00;          // timer/counter 2 reset
}
```

==========

5. ISR(INT1_vect)

- Rotary Encoder Switch가 눌리면 해당 Interrupt Service Routine이 실행된다.
- Switch 누른 횟수를 누적해서 짝수이면 재생이고 홀수이면 정지이다.
- WAV 음악 파일을 재생하기 위해서는 SD카드에서 Music_name[music_cnt] SD.open을 통해 접근한다. 접근한 파일에 music_info 함수를 통해 header 부분의 음악 정보를 알아내고 음악을 재생하기 위한 sr, bps, nc 값을 업데이트한다. 음악이 재생하는 도중에는 음악을 변경할 수 없도록 INT3 External Interrupt를 disable한다. 그리고 Timer_Counter interrupt1을 enable하여 음악이 재생한다.
- WAV 음악 파일을 정지하기 위해서는 먼저 cli()를 통해 모든 인터럽트 발생을 중지한다. 그리고 Timer_Counter interrupt1을 disable하여 음악이 정지한다. 음악이 정지가 되어있는 도중에는 변경이 가능하므로 INT3 External Interrupt를 enable한다. 여러 개의 파일을 동시에 열수 없으므로 현재 열려 있는 파일을 close() 함수를 통해 파일을 닫는다.

```
=================================================================

ISR(INT1_vect)                    // set up External ISR(interrupt service
routine)
{
  if (buttoncnt%2==0){
    lcd.setCursor(0,1);
    lcd.print("PLAYING");   // display "PLAY" on LCD when playing a music.

    // open the saved music file whose name is 'Music_name[music_cnt]' in
printDirectory(File dir) function
    myFile= SD.open(Music_name[music_cnt]);

    music_info(myFile);     // get the header information of the music

    EIMSK &= ~_BV(INT3);
    // disable interruption due to iNT3; no changing the music during playing
    TIMSK1 |= _BV(OCIE1A);  // timer/counter interrupt 1 enable, play the music
    }

  else{
    cli();                  // disable Timer/Counter1 interrupt
    myFile.close();
    lcd.setCursor(0,1);
    lcd.print("STOPPED");   // display "STOPPED" on LCD when music is stopped
    EIMSK |= _BV(INT3);
    // Enable interruption due to iNT3; only can change the music when music is
stopped
    TIMSK1 &= ~_BV(OCIE1A); // timer/counter 1 disable on stopped
    }
    buttoncnt++;
    sei();                  // enable Timer/Counter1 interrupt
}


=================================================================
```

6. ISR(INT3_vect)

- Rotary Encoder 회전하면 해당 Interrupt Service Routine이 실행된다.
- 왼쪽으로 회전하면 PhaseB pin에 0 신호가 전달되며 다음 WAV 음악 파일에 접근한다. 만약 다음 음악 파일이 존재하지 않다면 현재 음악 파일에 고정된다.
- 오른쪽으로 회전하면 PhaseB pin에 1 신호가 전달되며 이전 WAV 음악 파일에 접근한다.
- LCD, Serial 모니터에 해당 음악 파일 이름을 표시하며 음악을 처음부터 재생하기 위해 count=0으로 초기화한다.

```
=================================================================

ISR(INT3_vect)                    // set up External ISR(interrupt service
routine)
{
```

```
  cli(); //disable Timer/Counter1 interrupt
  if(digitalRead(PhaseB)) {
    music_cnt--;
  }
  else{
    music_cnt++;
  }

  if(music_cnt < 0){
    music_cnt = music_num-1;
  }
  if(music_cnt > music_num-1){
    music_cnt = 0;
  }

  lcd.clear();

  Serial.println(Music_name[music_cnt]);  // display on serial monitor
  lcd.setCursor(0,0);
  lcd.print(Music_name[music_cnt]);       // display on LCD monitor

  lcd.setCursor(0,1);
  lcd.print("STOPPED");
  count=0;  // to read data from the beginning
  sei();    // enable Timer/Counter1 interrupt
}
```

==========

7. ISR(TIMER1_COMPA_vect)

- Timer/Counter1에서 Compare Match가 발생할 때마다 해당 Interrupt Service Routine이 실행된다. 해당 Interrupt가 발생하는 Frequency는 SampleRate와 관련되어 있으며 OCR1A 값에 따라 변한다.
- WAV 음악 파일이 16bit-Stereo일 경우 재생하기 위해 4가지 조작이 필요하다. 가장 먼저 Little Endian으로 구성되어 있는 데이터를 Shift와 Bitwise를 통해 Weight에 맞게 Scale한다. 다음으로 16bit signed data에 0x8000을 더해서 16bit unsigned data로 변환한다. 또한 Stereo 경우 좌우 스피커에 알맞게 데이터를 입력해야 하므로 앞의 16bit unsigned data는 왼쪽 스피커, 뒤의 16bit unsigned data는 오른쪽 스피커에 입력한다. 마지막으로 16bit unsigned data를 8bit 두개로 나누어 PWM_H(OCnA), PWM_L(OCnB)에 입력한다.
- WAV 음악 파일이 16bit-Mono일 경우 16bit-Stereo일 경우와 한 가지 차이점이 있다. Mono일 경우 좌우 스피커에 같은 데이터가 입력되므로 16bit-Stereo와 다르게 (OC2A, OC2B) = (OC4A, OC4B)와 같이 좌우 스피커에 같은 데이터를 입력한다.
- WAV 음악 파일이 8bit-Stereo일 경우 재생하기 위해서 3가지 조작이 필요하다. 먼저 16bit data와 다르게 8bit data는 unsigned이다. 그러므로 부호를 바꾸는 연산이 필요하지 않다. 다음으로 좌우 스피커에 알맞게 앞의 8bit 데이터는 왼쪽 스피커, 뒤의 8bit data는

오른쪽 스피커에 입력한다. 마지막으로 좌우 각각의 8bit data는 PWM_L에 입력되고 PWM_H에는 0을 입력한다.

- WAV 음악 파일이 8bit-Mono일 경우 16bit-Mono와16bit-Stereo 차이와 같이 8bit-Stereo와 다르게 좌우 스피커에 같은 데이터를 입력한다.

==========

```
ISR(TIMER1_COMPA_vect)            // set up Timer ISR(interrupt service
routine)
{
  if(bps==16&&nc==2){     // 16bit&stereo WAV file

    uint16_t data3=0x0000;
    uint16_t data4=0x0000;
    data3 |= (buffer[i][count] | buffer[i][count+1]<<8);
    data4 |= (buffer[i][count+2] | buffer[i][count+3]<<8);
    data3= data3+0x8000;
    data4= data4+0x8000;
    OCR2A= data3>>8;
    OCR2B= data3 & 0xff;
    OCR4A= data4 >>8;
    OCR4B= data4 & 0xff;

    count= count+4;

    if(count >= 3000 &&b==0) count=0, i=1, b=1;
    else if(count>=3000&&b==1) count=0, i=0, b=0;
  }

  else if(bps==16&&nc==1){ // 16bit&mono WAV file

    uint16_t data3=0x0000;
    data3 |= (buffer[i][count] | buffer[i][count+1]<<8);
    data3= data3+0x8000;
    OCR2A= data3>>8;
    OCR2B= data3 & 0xff;
    OCR4A= OCR2A;
    OCR4B= OCR2B;

    count= count+2;
    if(count >= 3000 &&b==0) count=0, i=1, b=1;
    else if(count>=3000&&b==1) count=0, i=0, b=0;

  }

  else if(bps==8&&nc==2){ // 8bit&stereo WAV file

    OCR2A = 0;
    OCR2B = buffer[i][count];
    count ++;

    OCR4A = 0;
    OCR4B = buffer[i][count];
```

```
    count++;

    if(count>=3000&&b==0) count=0, i=1, b=1;
    else if(count>=3000&&b==1) count=0, i=0, b=0;

  }

  else {                  // 8bit&mono WAV file

    OCR2A=0;
    OCR2B=buffer[i][count];

    OCR4A=0;
    OCR4B=OCR2B;

    count ++;
    if(count>=3000&&b==0) count=0, i=1, b=1;
    else if(count>=3000&&b==1) count=0, i=0, b=0;

  }

}
```

8. void loop()

- 2가지 External Interrupt Service Routine을 통해 접근한 WAV 음악 파일의 데이터를 반복적을 읽어 buffer에 저장한다. 한 번에 3000byte를 읽어와서 데이터가 지연되어 재생이 안되는 현상을 방지했다.

```
void loop()
{

    myFile.read(buffer[0],3000);
    myFile.read(buffer[1],3000);

}
```

5. Trial and Error

1. void loop()에서 데이터를 읽는 속도에 따른 재생 오류

- 처음 알고리즘을 작성할 때는 loop()함수에서 myFile.read()를 통해 1byte씩 읽어와서 PWM 폭을 결정하는 OCnA, OCnB 값을 업데이트하는 과정을 모두 수행했다. 결과는 8bit WAV 파일은 매우 느리게 재생되고 16bit WAV 파일은 재생이 안됐다.
- 구글링과 선배에게 조언을 구하는 과정에서 해답을 얻을 수 있었다. Loop()함수에서 데이터를 1byte씩 읽고 OCnA, OCnB 값을 업데이트하는 과정을 모두 수행하게 되면 데이터를 읽어오는 속도가 느려 22.05 or 44.1KHz 음악을 재생할 수 없다.

- 그래서 loop() 함수에서는 WAV 음악 파일 데이터를 읽는 것만 반복하고 OCnA, OCnB 값을 업데이트하는 과정은 Timer Interrupt Service Routine1에서 수행하도록 했다. 그리고 데이터를 3000byte씩 읽어 데이터를 읽는 시간을 단축할 수 있었다.

2. WAV 음악 파일 음량에 따른 재생 오류

- WAV 음악 파일의 형태가 복잡해질수록 Perpheral board의 LCD Monitor 화면이 꺼지고 음악이 재생되는 도중에 꺼지는 빈도가 잦아졌다.
- 구글링을 통해 원인이 Peripheral Board에서 전류를 감당하지 못하기 때문인 것을 발견했고 음량을 줄인 WAV 음악 파일을 사용함으로써 해결했다.

2. Line Tracer

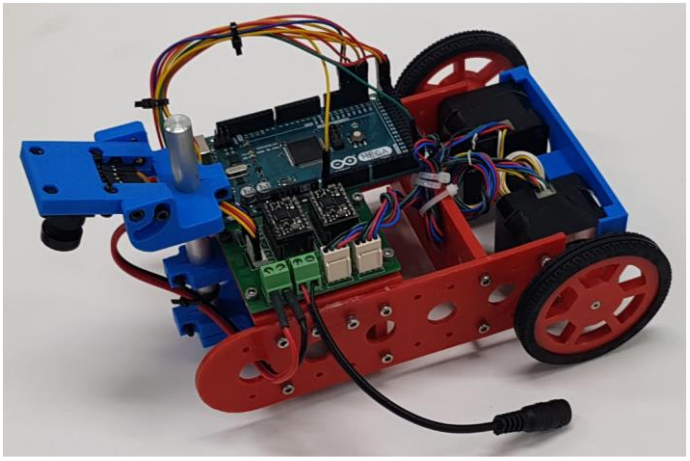
1. Abstract

임베디드 시스템 설계 두 번째 프로젝트로 Step Motor 로봇을 이용한 Line Tracer 구현이다. Robot 전면부에 부착되어 있는 Line Scan Camera에서 얻는 데이터를 바탕으로 좌우 모터의 속도를 조절하여 종방향 횡방향 제어를 한다. PID Control을 이용하여 일정한 종방향 속도에서 Desired Robot Position과 Current Robot Position을 Error로 설정하고 출력을 오른쪽 바퀴 속도에 더했다. 빠른 속도로 라인을 따라가는 것이 목표인 프로젝트의 목적에 맞게 종방향 속도를 증가시키면서 P, I, D Gain을 조절했다.

2. Hardware

1. Robot

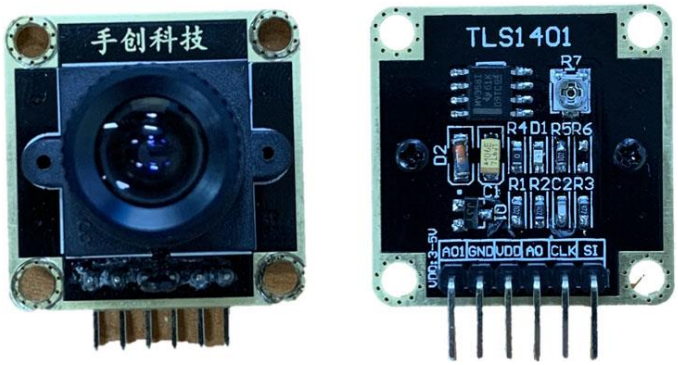
- Step Motor로 구동된다. Step Motor는 brush가 없는 DC모터로 회전 한번을 동일한 수의 단계로 나눈다. Microcontroller에서 Pulse 신호를 보내면 모터에서 unit step angle이 회전하고 본 프로젝트에서 사용하는 Step Motor의 Unit step angle은 0.225도이다.
- Robot은 이륜 Robot이다. 그러므로 두개의 바퀴의 속도 차이로 방향을 제어한다. 차량 직선 속도 등식은 $\frac{Wl+Wr}{2}$이며 회전 속도 등식은 $\frac{Wr-Wl}{2}$이다.



<FIG.5 Line Tracer Robot>

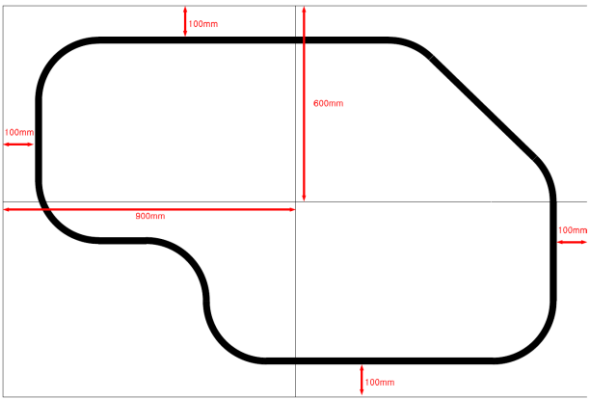
2. Line Scan Camera

- 밝기를 측정하는 센서로 128 x 1 배열의 ADC 값의 변화를 통해 Line의 위치를 판단할 수 있다.



<FIG.6 Line Scan Camera>

3. Map

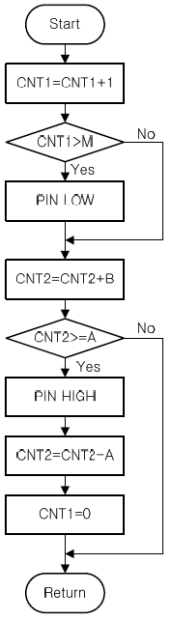


<FIG.7 Line Map>

3. Arudino Software

1. DDA(Digital Differential Analyzer)

- 원하는 방향과 속도로 Robot이 구동하기 위해서는 두 개의 Step Motor에 적절한 Frequency의 Pulse Signal을 주어야 한다. 본 프로젝트에서 사용하는 Robot은 이륜 Robot이므로 두 개의 Step Motor에 Pulse 신호를 각각 다르게 주었다.
- 본 프로젝트에서는 DDA 알고리즘을 사용했다. Carrier Frequency는 Timer/Counter Interrupt Frequency인 50KHz이며 Desired Frequency는 PID 제어기의 출력에 따라 변하도록 설정했다.



$$\frac{f_c}{f_d} = \frac{20000}{300} = \frac{10}{3} = \frac{A}{B} \qquad M = \text{int}\left(\frac{A}{B}\right) >> 1 = 1$$

Carrier clock	CNT1	CNT2	Pin value
1	1	3	LOW
2	2	6	LOW
3	3	9	LOW
4	4→0	12 → 2	HIGH
5	1	5	HIGH
6	2	8	LOW
7	3→0	11 → 1	HIGH
8	1	4	HIGH
9	2	7	LOW
10	3→0	10 → 0	HIGH
11	1	3	HIGH
12	2	6	LOW
13	3	9	LOW
14	4→0	12 → 2	HIGH
15	1	5	HIGH
16	2	8	LOW

<FIG.8 Example of DDA>

2. Camera Data 이진화를 통한 Robot Position 파악

- 본 프로젝트에서 사용하는 Camera에서 밝기에 따른 데이터를 ADC를 통해 0~1023의 크기로 받는다.
- Map을 제작하고 Camera test를 통해 값을 확인하니 주변 환경에 따라 변동이 큰 것을 확인했다. 그래서 많은 테스트를 통해 적절한 Threshold 값을 찾았고 흰 색 바탕일 경우는 1, 검정색 선일 경우 0으로 이진화를 했다.
- 1 X 128 배열 값을 차례대로 읽으며 1에서 0으로 변하는 위치를 Left_Edge로 설정했고 0에서 1로 변하는 위치를 Right_Edge로 설정했다. 그리고 두 값의 중간 값을 현재 Robot

의 Position으로 설정했다.

3. PID Control

- Line의 중앙값과 현재 Position 차이에 따른 이륜 모터 속도 제어를 위해 PID Control을 사용했다.
- Input은 Desired Robot position인 Line의 중앙값이며 출력을 오른쪽 Motor에 해당하는 Pulse Frequency에 더했다.
- P Gain 조정을 통해 반응 속도를 증가했고 I, D Gain 조정을 통해 Overshoot와 Constant Error을 줄였다.

3. Algorithm

1. void ADC_Get(unsigned char num)

- Line Scan Camera에서 측정한 데이터를 ADC Pin을 통해 받고 저장하는 함수이다.
- 변환 중에는 ADSC 값이 1이며 변환이 끝나면 0으로 변한다.
- 데이터는 0~1023의 값을 가진다. 변환이 끝나면 camera_data[num] 값에 저장한다.

```
=================================================================
void ADC_Get(unsigned char num)   // read camera analog data through ADC pin
{
  ADCSRA |= (1<<ADSC); // Start conversion
  while(ADCSRA & (1<<ADSC));
  camera_data[num] = ADCW;
}
=================================================================
```

2. void DDA_parm(int WL, int WR)

- DDA 알고리즘에 사용할 변수를 설정하는 함수이다.
- Carrier Frequency는 Timer/Counter Interruption Frequency인 50KHz이다.
- B[0],M[0]은 왼쪽 Step Motor에 전달되는 Pulse Frequency를 결정하는데 사용되며 B[1],M[1]은 오른쪽 Step Motor에 전달되는 Pulse Frequency를 결정하는데 사용된다.

```
=================================================================
void DDA_parm(int WL, int WR)     // set the parameters used in DDA algorithm
{
  TCNT1 = 0;
  B[0] = WL+0.0001;
```

```
  M[0] = int(50000 / B[0]) >> 1;
  B[1] = WR+0.0001;
  M[1] = int(50000 / B[1]) >> 1;
}
```

==========

3. ISR(TIMER1_COMPA_vect)

- DDA 알고리즘이 수행되는 ISR(Interrupt Service Routine)이다.
- 좌우 Step Motor 모두에 대해 두번의 알고리즘이 수행된다.
- Carrier Frequency는 해당 ISR의 Frequency인 50KHz이다

==========

```
ISR(TIMER1_COMPA_vect)          // set up timer ISR(interrupt service
routine)
{
    for (int i = 0; i < 2; i++)
    {
        cnt1[i]++;
    }
    for (int i = 0; i < 2; i++)
    {
        if (cnt1[i] > M[i])
        {
            digitalWrite(left_motor,0);
            digitalWrite(right_motor,0);
        }

        cnt2[i] += B[i];

        if (cnt2[i] >= 50000)
        {

            digitalWrite(left_motor,1);
            digitalWrite(right_motor,1);
            cnt2[i] -= 50000;

            cnt1[i] = 0;
        }
    }
}
```

==========

4. void PIDcalc()

- 좌우 Step Motor의 회전 속도를 PID 제어를 통해 조절하는 함수이다.
- PID 제어의 입력은 Desired Robot Position으로 라인의 중간이다.
- PID 제어의 출력을 왼쪽 바퀴 속도에 더해서 Desired Robot Position으로 Robot이 향하게 한다.

- 차량의 속도를 높이면서 Desired Robot Position에 도달하는 속도 또한 높여야 하므로 P gain을 높게 설정했다. 그리고 P gain이 증가하면서 Robot이 불안정하게 overshoot가 발생하는 것을 확인했고 I gain을 조금씩 증가시켜 overshoot를 줄였다. 마지막으로 가끔씩 코너 부근에서 이탈하는 경우가 발생하였는데 D gain을 조금씩 증가시키면서 이탈하는 빈도를 줄였다.

- 많은 시도 끝에 최종적으로 (P Gain, I Gain D Gain) = (32, 3, 0.5)으로 결정했고 PIDInit() 함수에서 설정했다.

==========

```
void PIDcalc()
// set the rotation velocity of motor to locate robot at desired position
{
  PID.Err = PID.Ref - PID.state;
   // Error calculation
  PID.Out_tmp = PID.Kp * PID.Err + PID.Ki * PID.Intg + PID.Kd * PID.Deriv;
  // PID Output

  if (PID.Out_tmp > PID.Max) PID.Out = PID.Max;
  else if (PID.Out_tmp < PID.Min) PID.Out = PID.Min;
  else PID.Out = PID.Out_tmp;

  PID.SatErr = PID.Out - PID.Out_tmp;  //Anti -windup

  PID.Intg += PID.samT * (PID.Err + PID.Kc * PID.SatErr);
  // Error input to the Integral controller.
  PID.Deriv = (PID.Err - PID.Err_past) / PID.samT;
  //Error input to the Derivative controller.

  PID.Err_past = PID.Err;
}
```

==========

5. void loop()

- 현재 Robot의 위치를 알기 위해 Line Scan Camera data를 이진화를 한 뒤 PID Control에 입력하여 좌우 Step Motor의 새로운 속도를 설정하는 함수이다

- Robot을 구동할 Map을 제작한 뒤 Line Scan Camera를 통해 ADC pin의 Decimal Data를 확인한다. 경계 부근의 값을 확인하고 Threshold 값을 결정한다. 그리고 Threshold 값을 기준으로 Decimal Data를 Binary Data로 변환한다. 흰색 바탕은 1 검정색 선은 0으로 변환한다.

- Binary Data로 변환된 데이터에서 1에서 0으로 변하는 곳을 Left_Edge, 0에서 1로 변하는 곳을 Right_Edge로 설정하고 현재 Robot의 위치는 두개의 Edge 값의 중앙값으로 설정한다.

➢ PID제어기의 Error를 Desired Robot Position과 Current Robot Position의 차이로 설정하고 출력 값을 오른쪽 Step Motor의 회전 속도에 더하여 Error를 0으로 변하도록 한다.

==========

```
void loop()
{
    int i;

    // initialize reading camera data
    PORTA |= (1 << SI);   // SI : HIGH
    PORTA |= (1 << CLK);  // CLK : HIGH
    PORTA &= ~(1 << SI);  // SI : LOW

    delay(1);

    for (i = 0; i < 129; i++)
  {
    ADC_Get(i);           // read data input of A0 pin
    PORTA &= ~(1 << CLK); // CLK low
    PORTA |= (1 << CLK);  // CLK high

   if (camera_data[i] > scan_TH) {
     camera_data[i] = 1;
    }
    else {
     camera_data[i] = 0;
    }

    if(i>0)
    {
      if(camera_data[i-1]^camera_data[i]) {
        if(camera_data[i]==0) left_edge=i;
        else right_edge=i;
      }
    }
  }

    PORTA &= ~(1 << CLK);  //CLK low
    Position= (float)(right_edge+left_edge)/2; // current position of the robot

    PID.Ref=64; // desired position of robot is middle of the line
    PID.state=Position;

    PIDcalc();

    con_freq=PID.Out;
    WR= con_freq+vel;
    WL=(2*vel)-WR;

    DDA_parm(WL,WR);    // update DDA parameters using updated WR,WL

    while (!((time_out - micros()) & 0x80000000));
    time_out += MicrosSampleTime;
```

}
==========

4. Trial and Error

1. Decimal Line Scan Data 사용

- ADC Pin을 통해 입력 받은 Line Scan Data를 이진화 과정 없이 10진수 값을 그대로 사용했다. 10진수의 값은 2진수 값과 다르게 외부 환경에 따라 변동이 커서 현재 Robot의 위치가 정확하지 않았다.
- PID 제어기에 정확하지 않은 현재 위치를 입력하면 Robot이 Line일 제대로 따라가지 못했지만 이진화를 통한 정확한 현재 위치 입력을 통해 문제를 해결했다.

2. PID 제어

- 처음 구상했던 모터 제어는 PID제어 없이 좌우 Step Motor에 일정한 Frequency의 Pulse를 입력하여 현재 Robot의 위치에 따라 바퀴 하나만 구동하도록 했다.
- Robot의 속도가 매우 느렸고 Line을 제대로 따라가지 못했다.
- Error를 0으로 하는 PID 제어에서 Error가 Desired Robot Position과 Current Robot Position의 차이로 설정할 수 있다는 것을 알아냈고 적용함으로써 정확성과 속도를 증가할 수 있었다.