**마이크로프로세서설계및실습**

**최종 결과 보고서**

**-피아노와 음악 시각화 장치-**

**서울시립대학교**

**컴퓨터과학부**

**2017920002 강민성**

**2017920040 원우인**

1. **역할 소개**

원우인 – 전자 피아노를 만들고 곡을 재생하는 mp3를 만드는 역할입니다. 실습 때 배운 스위치와 인터럽트, 부저 지식으로만 코드를 생성하였고 E 포트에 존재하는 4번 스위치에 모드 변경 역할을 부여하여 노래 재생 모드, 피아노 연주 모드 2개로 구성하였습니다. 노래가 재생 중일 때는 몇 번 곡인지를 JKIT-128-1의 FND를 통해 표현하고 피아노 연주 모드일 때는 도,레,미,파,솔,라,시,도 각각 빵판의 스위치 하나, JKIT-128-1의 led 하나와 대응되도록 구성하였습니다. 피아노 연주 시에는 polling 방식을 사용하여 연주합니다.

강민성 – 음향신호를 감지하고 시각적으로 나타내는 장치를 만드는 역할입니다. 음향 입력 센서와 ATmega128의 ADC를 통해 음향 신호를 입력 받고 푸리에 변환을 통해 각 주파수 별 신호의 크기를 도트 매트릭스에 나타냅니다. 푸리에 변환은 기 작성된 라이브러리를 사용하며 도트 매트릭스는 A포트와 C포트, 음향 센서는 ADC를 지원하는 F포트에 연결해 작동시켰습니다.

1. **주제 및 기능 소개**

- 음성 센서와 ADC를 통해 아날로그 신호인 음성을 입력 받고 푸리에 변환을 통해 변환, 주파수별로 분해합니다 이를 8개 단위로 나눠 평균치를 저장합니다.

- 분해된 주파수별의 신호세기를 도트 매트릭스에서 표현합니다. 주파수가 갖는 진폭이 높을 수록 도트 매트릭스 상에서 높은 막대로 나타납니다. 좌에서 우로 갈수록 저주파에서 고주파를 나타냅니다.

- 1번 스위치(PE4)를 통해 연주 모드로 바꾼 후 존재하는 8개 건반(스위치)을 누르면 해당하는 음계가 재생되는 피아노 기능이 존재하고 음계에 해당하는 led 불빛도 들어옵니다.

- 1번 스위치(PE4)를 통해 음악 재생 모드로 바꾼 후 2번 스위치(PE5)를 통해 존재하는 4가지 노래(크리스마스 노래 등)를 감상 가능합니다.

1. **진행 내용**

전자 피아노의 경우 연주 모드와 노래 재생 모드를 jkit-128-1의 e포트와 연결된 1번 스위치(PE4)를 누르면 하강 엣지 인터럽트를 통해 바꿀 수 있도록 하였습니다. 노래 재생 모드일 경우 c포트와 연결된 FND에서 노래의 번호가 나오고 e포트와 연결된 2번 스위치(PE5)를 누르면 하강 엣지 인터럽트를 통해 노래를 바꿀 수 있도록 하였습니다.

피아노 연주 모드일 경우에 빵판에 스위치 8개를 연결하여 각각 도,레,미,파,솔,라,시,도 음계에 대응되도록 하였으며 polling 방식을 통해 스위치 하나가 눌러지면 해당 스위치와 대응되는 음계 클록 사이클 값이 TCNT0에 들어가는 방식으로 구현하였습니다. 또, 각 스위치 8개와 A포트의 led 8개가 대응되어 음계 하나를 연주할 시 led 하나가 깜빡거리는 식으로 구현하였습니다.

음향 시각화 장치의 경우 먼저 ADC(analog to digital convert)기능의 활성화를 위해 레지스터를 적절히 초기화 해주었습니다. PORT F0 핀의 사용을 위해 ADMUX에 0x00의 값을 사용했습니다. ADC 제어 레지스터에서 ADEN 비트를 켜 ADC를 활성화하고 ADSC, ADFR, ADIE 등의 비트를 켜 프리 러닝 모드로 작동시켰습니다. 다음으로 인터럽트 서비스 루틴을 작성해 각 루프 초기에 인터럽트를 통해 음성신호를 입력 받고 fft 라이브러리에 포함된 fft\_input, fft\_execute, fft\_output 함수를 차례로 호출해 주파수 별의 신호로 분해한 뒤 8개 단위로 평균값을 내어 도트 매트릭스에 표시하기 위한 부울 값의 2차원 배열로 저장하고 출력했습니다.

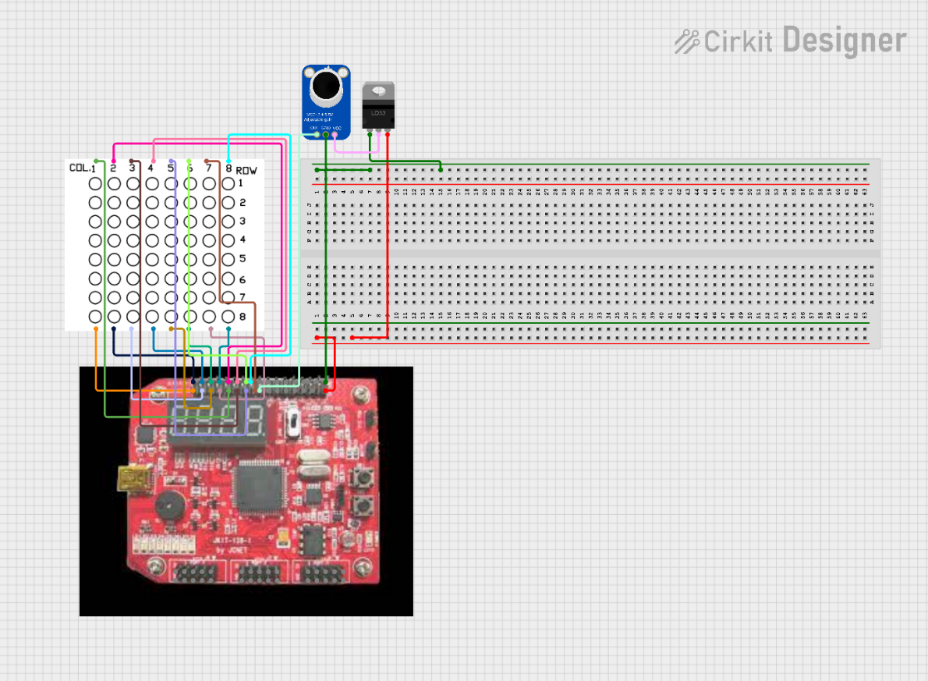
1. **추가 부품 사용 내역**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| No. | 제품명 | 상품코드 | 단가 | 수량 | 합계 |
| 1 | JKIT-128-1 | 1059759 | 54,000 | 1 | 54,000 |

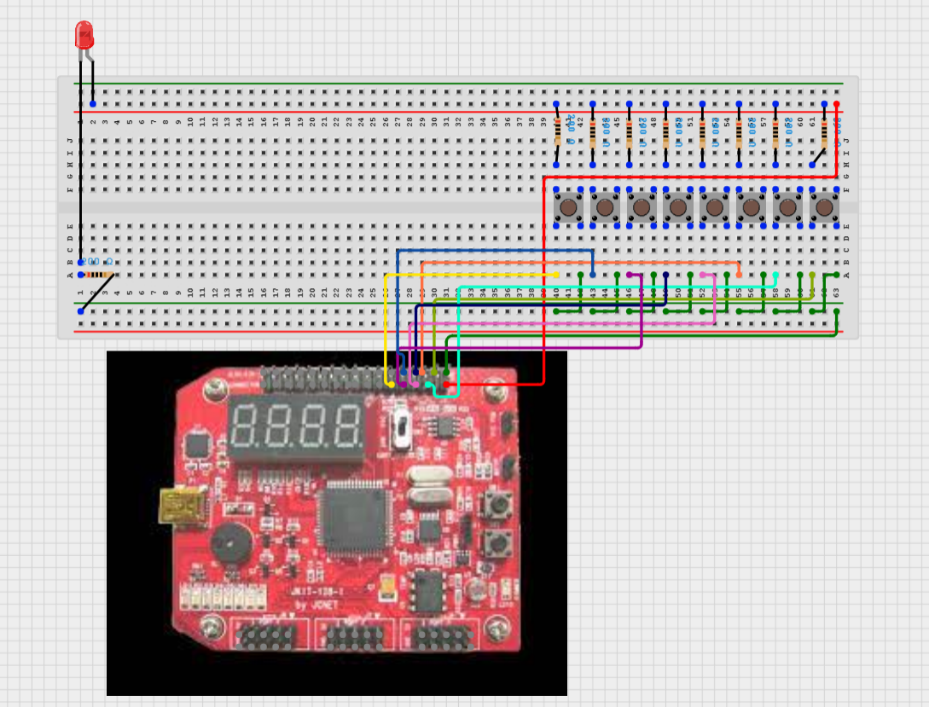
* Pin 수가 부족할 것을 고려하여 Jkit-128-1을 2개 사용하여 제작

1. **회로도**

(1)오디오 스펙트럼 구현



(2) 전자 피아노와 mp3 구현



1. **소스코드**

|  |
| --- |
| //오디오 스펙트럼 코드  #include<stdint.h>  #include"ffft.h"  #include<math.h>  #include<util/delay.h>  #include<avr/io.h>  #include<avr/interrupt.h>  #include<stdio.h>  #include<stdbool.h>  #include<avr/pgmspace.h>  #define *F\_CPU* 16000000UL  #define byte unsigned char  #define ADC\_CHANNEL 0  byte displayData[8][8];    *int16\_t* capture[FFT\_N];  complex\_t bfly\_buff[FFT\_N];  *uint16\_t* spectrum[FFT\_N/2];  volatile byte samplePos = 0;    void setup()  {  DDRA = 0xFF;  DDRC = 0XFF;  PORTA= 0x00;  PORTC= 0xff;  ADMUX = ADC\_CHANNEL;  ADCSRA = (1 << ADEN) |  (1 << ADSC) |  (1 << ADFR)|  (1 << ADIE) |  (1 << ADPS2)|(1<<ADPS1)|(1<<ADPS0);  sei();  }  void loop() {  while(ADCSRA & (1<<ADIE))    fft\_input(capture, bfly\_buff);  samplePos = 0;  ADCSRA |= (1 << ADIE);  fft\_execute(bfly\_buff);  fft\_output(bfly\_buff, spectrum);  int i, j, avg;  for(i=0;i<8;i++)  {  avg = 0;  for(j=0;j<8;j++)  {  avg += spectrum[j+i\*8];  }  avg /= 8;  if(avg/(8-i) > 2)  {  if(i==0)  avg /= 20;  for(j=0;j<avg/(8-i);j++)  {  displayData[i][j] = 1;  }  }  }    for(i=0;i<8;i++)  {  PORTA = 0;  for(j=0;j<8;j++)  {  PORTA |= (displayData[7-i][j] << j);  }  PORTC = (0xff)^((0x01 << i) % (0xff));  *\_delay\_ms*(2);  }    for(i=0;i<8;i++)  {  for(j=0;j<8;j++)  {  displayData[i][j] = 0;  }  }  }  ISR (ADC\_vect) {  static const *int16\_t* noiseThreshold = 4;  *int16\_t* sample = ADC;    capture[samplePos] = ((sample > (512-noiseThreshold)) &&  (sample < (512+ noiseThreshold))) ? 0 : sample - 512;    if(++samplePos >= FFT\_N) ADCSRA &= ~(1 << ADIE);    }  int main()  {  setup();  while(1)  {  loop();  }  } |

|  |
| --- |
| // 전자 피아노와 mp3 구현 코드  #include <avr/io.h>  #include <avr/interrupt.h>  #include <stdio.h>  #include <stdbool.h>  #define F\_CPU 16000000UL  #include <util/delay.h> // 무조건 F\_CPU 뒤에 위치  #define ON 0  #define OFF 1  #define PLAY 0  #define PERFORM 1  #define DO\_data 17  volatile int state = OFF;  volatile int tone;  volatile int play = PLAY;  #define DO 0  #define DOSHARP 1  #define RE 2  #define MIFLAT 3  #define MI 4  #define FA 5  #define FASHARP 6  #define SOL 7  #define SOLSHARP 8  #define RA 9  #define SIFLAT 10  #define SI 11  #define DDO 12  #define RRE 13  #define NON 14  #define EOS -1  char f\_table[15] = {17,30, 43, 55, 66, 77, 87, 97, 105 ,114, 122, 130, 137, 150, 255};  int song1[] = {SOL, MI, MI, SOL, MI,DO, RE, MI, RE, DO, MI, SOL, DDO,SOL, DDO, SOL, DDO, SOL, MI, SOL,RE, FA, MI, RE, DO, EOS};  int song2[] = {RE,SOL,SOL,RA,SOL,FASHARP,MI,MI,MI,RA,SI,RA,SOL,FASHARP,RE,RE,SI,DDO,SI,RA,SOL,MI,RE,RE,MI,RA,FASHARP,SOL,EOS};  int song3[] = {SI,RE,MI,SOL,SOL,SI,RE,MI,SI,SOL,SI,RRE,SI,RRE,SI,RA,RA,SOL,RE,SOL,FASHARP,SOL,SOL,FASHARP,RA,SOL,EOS};  int song4[] = {FASHARP,FASHARP,MI,MI,MIFLAT,FASHARP,MIFLAT,DOSHARP,DOSHARP,MIFLAT,DOSHARP,SI,SI,DOSHARP,MIFLAT,DOSHARP,DOSHARP,DOSHARP,SI,MIFLAT,DOSHARP,SI,SOLSHARP,SI,DOSHARP,MIFLAT,DOSHARP,DOSHARP,DOSHARP,EOS};  unsigned char digit[10] = {0x3f, 0x06,0x5b, 0x4f, 0x66, 0x6d, 0x7c, 0x07,0x7f, 0x67};  unsigned char led[9] = {0x00, 0x80, 0x40,0x20, 0x10, 0x08, 0x04, 0x02, 0x01};  unsigned char compare[9] = {0x00, 0x01, 0x02,0x04, 0x08, 0x10, 0x20, 0x40, 0x80};  int piano[8] = {0,2,4,5,7,9,11,12};  int piano2[9] = {EOS,DO,RE,MI,FA,SOL,RA,SI,DDO};  volatile int whatsong = 0;  volatile int pianosong = 0;  //volatile int j=0;  volatile int i=0;  ISR(INT4\_vect){  if (play == PLAY)  {  pianosong = 0;  PORTA = led[pianosong];  PORTC = 0x00;  PORTD = 0x00;  PORTF = 0x80;  play = PERFORM;  \_delay\_ms(10);  }  else  {  PORTA = 0x00;  play = PLAY;  PORTC = digit[whatsong+1];  PORTF = 0x00;  \_delay\_ms(10);  }  }  ISR(INT5\_vect){  if(play == PLAY){  whatsong = (whatsong+1)%4;  PORTC = digit[whatsong+1];  \_delay\_ms(10);  }  else{    /\*  j = (j+1)%8;  PORTA= led[j];  \_delay\_ms(10);  \*/    }  }  ISR(TIMER0\_OVF\_vect)  {  if (state == ON)  {  PORTB = 0x00;  state = OFF;  }  else  {  PORTB = 0x10;  state = ON;  }    if(play==PLAY){  TCNT0 = f\_table[tone];  }  else{  //TCNT0 = f\_table[piano[j]];  TCNT0 = f\_table[tone];  }  }  void playsong(){    switch(whatsong){  case 0:  do{  tone=song1[i++];  \_delay\_ms(500);  if(play==PERFORM){  i=0;  break;  }  if(whatsong!=0){  i=0;  break;  }  }while(tone!=EOS);  i=0;  break;    case 1:  do{  tone=song2[i++];  \_delay\_ms(500);  if(play==PERFORM){  i=0;  break;  }  if(whatsong!=1){  i=0;  break;  }  }while(tone!=EOS);  i=0;  break;    case 2:  do{  tone=song3[i++];  \_delay\_ms(500);  if(play==PERFORM){  i=0;  break;  }  if(whatsong!=2){  i=0;  break;  }  }while(tone!=EOS);  i=0;  break;    case 3:  do{  tone=song4[i++];  \_delay\_ms(500);  if(play==PERFORM){  i=0;  break;  }  if(whatsong!=3){  i=0;  break;  }  }while(tone!=EOS);  i=0;  break;  }    }  void piano\_play(){  int it = 0;  if((PIND & 0x80)==0) it =8;  if((PIND & 0x40)==0) it = 7;  if((PIND & 0x20)==0) it = 5;  if((PIND & 0x10)==0) it = 6;  if((PIND & 0x08)==0) it = 3;  if((PIND & 0x04)==0) it = 1;  if((PIND & 0x02)==0) it = 2;  if((PIND & 0x01)==0) it = 4;  if((PIND & 0xff)==0xff) it = 0;  PORTA = led[it];  tone = piano2[it];  }  int main(void)  {  DDRA = 0xff; // LED 출력  DDRB = 0x10; // 부저 출력  DDRD = 0x00;  DDRE = 0xcf; // 5,4번 pin 입력으로  DDRF = 0x80;  PORTF = 0x00;    DDRC = 0xff; // C 포트는 모두 출력  DDRG = 0xff; // G 포트도 4개는 출력  PORTC = digit[1]; // ‘0’ 표현  PORTG = 0x01;    TCCR0 = 0x03; // 8분주, 0.5us  TIMSK = 0x01; // Overflow  EICRB = 0x0a; // 하강 에지 1010, 인터럽트  EIMSK = 0x30; // INT5,INT4 interrupt enable  TCNT0 = f\_table[song1[i]]; // 해당 숫자~256까지가 지난 후(overflow) interrupt 발생  sei(); // interrupt enable  while(1){  if(play== PLAY){  playsong();  }  else{  piano\_play();    /\*  while(1){  piano\_play();    if(play==PLAY){  i=0;  break;  }    }  \*/    }  }  } |

1. **결과 및 고찰**

전자 피아노 연주 모드와 재생 모드 각각을 구별하기 위해 스위치 1번을 누를 시에 interrupt가 발생하여 정해둔 interrupt service routine 안에서 변수 play를 PLAY, PERFORM 2개 값으로 바꿔주면서 코드 내부적으로 구별하게 하였습니다. 외부적으로는 Breadboard에 F포트에서 나온 케이블과 연결된 led가 연주 모드일 시 켜지고 재생 모드일 시 꺼지도록 함으로써 구별하였습니다.

처음에 작동시킬 때는 재생 모드로 설정되어있고 재생 모드일 때는 스위치 2번을 누르면 interrupt가 걸려 해당 코드를 실행함으로써 연주되는 노래(song1 ~ song4, 배열 형태로 저장되어있음)를 바꿀 수 있습니다. 몇 번 노래를 틀고 있는지는 C포트에 연결된 FND로 나타내주었으며 원래 LCD로 노래 제목까지 보여줄 예정이었지만 구매한 케이블이 부족하여 LCD만 제외하게 되었습니다.

연주 모드에 사용할 전자 피아노를 구현하기 위해 스위치 8개를 구매하여 각각을 도,레,미,파,솔,라,시,도로 선정하여 한 쪽은 ground로, 한 쪽은 D포트로의 입력으로, 나머지 한 쪽은 5V로 연결하여 스위치를 누르면 5V와 GND가 연결되어 입력이 발생하고 이것이 확인되면 타이머/카운터 레지스터(TCNT0)에 음계마다의 클록 사이클 값을 넣어서 지연 시간 후에 overflow interrupt가 발생하도록 하여 부저로 음을 발생시킵니다. 또한, JKIT-128-1에 달려있는 LED 8개를 통해 각각 음계를 연주하면 대응되는 하나의 led가 켜지도록 설계하였습니다.

피아노 연주 시 건반에 대응되는 스위치가 접지 상태가 불량하여 두 번 정도 눌러야 소리가 나올 때가 존재하여 더 좋은 스위치를 썼으면 어땠을까하는 아쉬움이 존재합니다. 또, 본래 인터럽트 방식으로 피아노를 구현하려고 하였으나 interrupt에 사용할 수 있는 pin 수를 고려하여 polling 방식으로 선회하여 전자 피아노를 구현하였습니다. 마지막으로, 노래를 저장하여 재생해주는 부분에 있어서 박자까지 맞춰 넣어보려했으나 노래마다 음계가 재생되는 시간까지 고려하기에는 너무 힘들 것 같아 노래 배열 안의 음계 개수를 조절하여 박자를 맞추려고 노력하였습니다.

음향 시각화 시 노이즈 제거에 미흡해 실제론 입력되지 않는 특정 주파수의 막대가 계속해서 높게 나타나는 경향이 나타났습니다. 노이즈를 잡기위해 인터럽트 서비스 루틴에서 입력 받은 그대로 신호를 사용하지 않고 noise threshold를 정해 신호의 범위를 제한했지만 역부족이었던 것 같습니다.

음향 시각화 시 주파수 단위에 따라 올라오는 막대가 직접 음향을 들을 때와 일치하지 않는, 직관과 어긋나는 경향을 확인할 수 있었습니다. 이퀄라이저를 적절히 적용해 이를 해소할 수 있음을 알게 되었지만, 원리부터 파악해 이퀄라이저를 만들거나 디버깅 수단을 갖춰 꼼꼼히 확인하기에는 힘에 부쳤고, 실험해가며 적당하게만 조정한 점이 아쉬운 결과를 낳았다고 생각합니다.