AVR 프로세서를 이용한 MIDI 인터페이스 제작

정보통신공학전공　김지원
졸업작품 지도교수　이현창

제1장　작품의 개요

본 작품에서는 MIDI 프로토콜을 이용하여 컴퓨터와 피아노가 통신할 수 있는 MIDI 인터페이스를 제작하였다. 다이나믹 스캐닝을 이용해 피아노 건반을 프로세서에서 검출하였고, 직렬통신을 이용해 컴퓨터와 통신하도록 제작 하였다. PC를 총 두 대 이용하여 하나는 MIDI 프로토콜을 이용하여 피아노 연주 시 연주상황이 컴퓨터로 실시간 전달되어 음계 값이 악보에 표시되게 하였고, 다른 하나는 RS-232C를 이용해 피아노 건반 번호가 하이퍼터미널에 표시될 수 있도록 하였다. 다이나믹 스캐닝으로 인해 스위치 검출 시 발생하는 고스트 키 현상을 ADC를 이용해 제거하였으며, 여러 개의 건반을 동시에 처리하기 위해 스위치 상태마다 플래그를 선언하였다.

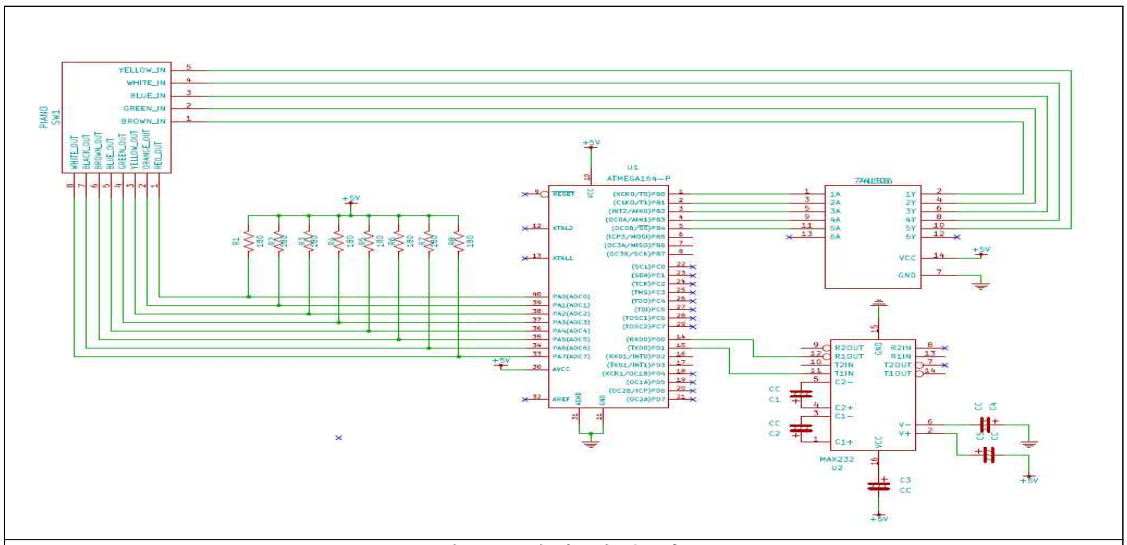
제2장　작품의 원리 및 구성

2-1 하드웨어

2-1-1. 다이나믹 스캐닝을 통한 피아노 스위치 검출

MIDI 인터페이스를 제작하기 위하여 총 32개 건반으로 이루어져 있는 피아노는 내부적으로 40개의 스위치로 구성되었다. 본 작품에서는 다이나믹 스캐닝 회로를 이용해 스위치 상태를 검출하도록 하였다. 아래 <그림 1>은 작품의 전체 회로도이다. ATmega164A 프로세서가 74LS06 칩의 오픈 컬렉터를 통해 순서대로 출력을 내보내고 스위치가 눌렸을 때 포트A에서 'L'이 검출되면서 스위치를 검출하게 된다. 다이나믹 스캐닝 방식은 소량의 프로세서 포트를 이용해 대량의 스위치들을 순차적으로 검출할 수 있는 장점을 가졌다. 하지만 하드웨어적 특징에 의해 특정 스위치들이 함께 눌리는 경우 다른 스위치가 함께 눌린 것으로 검출되는 고스트 키 현상이 발생한다는 단점이 있다.

스위치 상태는 ATmega164A의 포트A로 입력받아 검출하도록 하였다. 피아노의 신호선은 총 13개로 이루어졌는데 그 중 8개가 프로세서의 PA0부터 PA7까지 연결 되었다. 포트 A는 ADC 회로가 내장 되어 있는데 이를 이용해서 추후에 문제가 된 고스트 키 현상을 제거할 수 있도록 하였다. 각각의 선들에는 풀업 저항이 연결되어 건반 스위치가 눌리지 않았을 때 프로세서의 입력라인이 floating 되므로 이를 방지하기 위함이다. 나머지 신호선 5개는 포트B에 오픈컬렉터를 통해 'L'출력을 내보내어 스위치 상태를 검출하도록 하였다. 오픈 컬렉터(open collector) 는 트랜지스터의 컬렉터가 개방된 형태를 갖춘 구조의 출력 회로로 출력단에 5V를 넣지 않으면 제 3의 상태 즉, 0도 1도 아닌 상태가 된다. 오픈컬렉터를 사용하지 않으면, 출력 단에 5V가 출력되어 피아노 건반을 동시에 여러 개 누르게 될 때 프로세서가 파손될 우려가 있다.

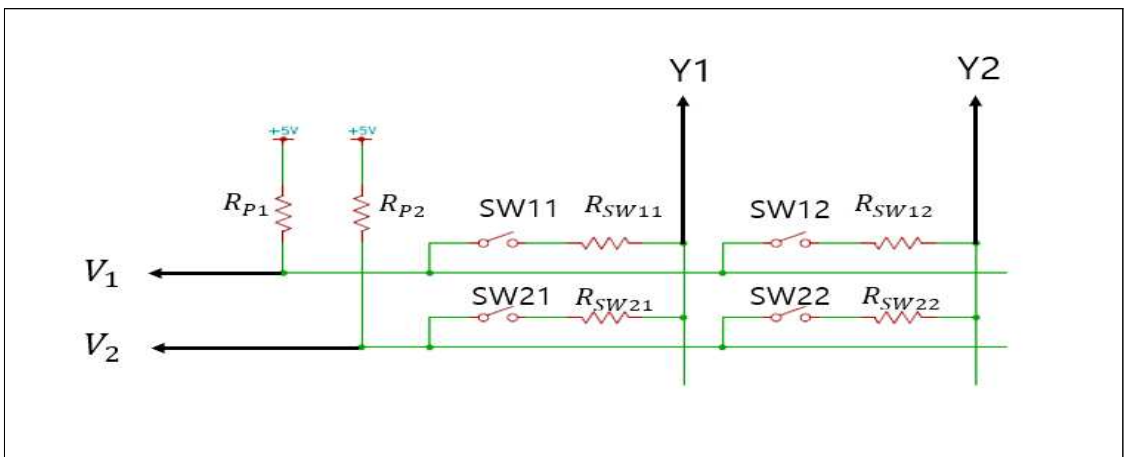


<그림 1> 전체 작품 회로도

2-1-2. 고스트 키 현상

고스트 키(Ghost key) 현상이란 3개 이상의 키를 동시에 눌렀을 때 사용자가 누르지 않은 다른 키가 눌린 것으로 검출되는 현상이다. 피아노 연주 시 건반이 3개 이상 누르는 경우가 생겨 고스트 키를 제거하지 않게 되면 문제가 발생한다.

아래 <그림 2>에서는 다이나믹 스캐닝을 이용한 피아노 내부 등가회로를 나타낸 것이다. 피아노 내부회로는 5x8 매트릭스로 이루어져있고, <그림 2>는 일부분을 나타낸 것이다. SW11, SW12, SW21, SW22는 피아노 건반 스위치이다. R_{SW11}, R_{SW12}, R_{SW21}, R_{SW22}은 각 건반의 고유의 저항 값이다. 각각의 스위치 저항 값은 100Ω에서 300Ω 사이로 측정되었는데 계산식에서는 200Ω으로 계산하였다. R_{P1}와 R_{P2}는 풀업저항으로 작품에서는 180Ω으로 하였다. 따라서 SW11, SW12, SW21이 동시에 눌렸을 때 SW22가 눌린 것으로 검출되는 되는데 이것을 고스트 키 현상이라고 한다.



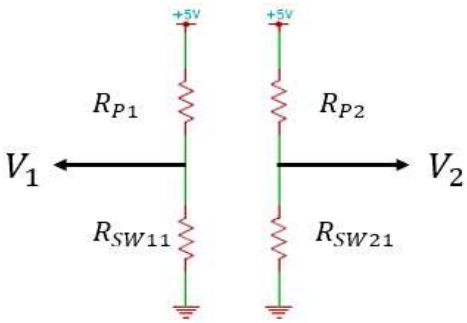
<그림 2> 다이나믹 스캐닝을 이용한 피아노 내부 등가회로

위의 <그림2>에서 정상적인 상태인 Y1열을 스캐닝하면 <그림 3>과 같은 등가회로가 구성되는데 이때 검출전압 V_1과 V_2는 동일한 값이 검출된다. 그러나 위의 <그림2>의 허상키가 발생하는 Y2열을 스캐닝하면 <그림 4>와 같은 등가회로가 구성되어 정상 키에 의한 전압 V_1과 허상 키에 의한 전압 V_2에 차이가 발생한다.. 따라서 V_1은 정상키가 눌렸을 때 전압이고, V_2는 고스트 키 가 검출되는 전압이다. 아래는 <그림 3>과 <그림4>를 통해 V_1과 V_2를 수식으로 나타낸 것이다. 이때 V_P는 5V이다.

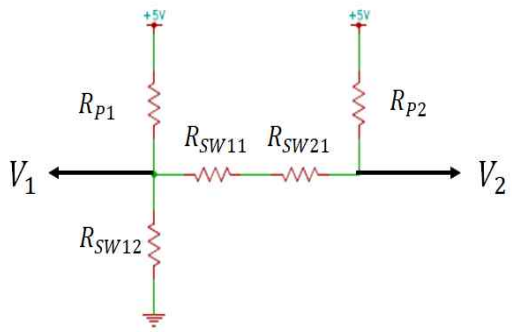
$$V_1 = \frac{R_{SW}}{R_{SW} + (R_P + 2R_{SW}) \| R_P} \cdot V_P$$

$$V_2 = \frac{2R_{SW}}{2R_{SW}^2 + R_P}(V_P - V_1) + V_1$$

수식을 통한 계산값은 V_1 = 2.964V V_2 = 4.368V 로 나타난다. 따라서 <그림 5> 에서는 정상키에 의한 전압이 3V에서 나타난 것을 볼 수 있다. <그림6> 에서는 고스트 키의 전압이 4.6V에서 나타난 것을 볼 수 있다



<그림 3> Y1을 스캐닝 했을 때 등가회로



<그림 4> 고스트 키가 발생할때의 등가회로

<그림 5> 정상키

<그림 6> 고스트 키

2-2 소프트웨어

2-2-1. 직렬통신

직렬통신(Serial Communication)은 여러 개의 비트 데이터를 한 가닥의 전선을 사용하여 차례로 전송하는 것을 말한다. 클록신호 전송 유무에 따라 동기식과 비동기식으로 나뉜다. 송신측에서 수신측으로 클록신호가 전달되면 동기통신이라 하고, 클록 신호가 전달되지 않으면 비동기 통신 이라 한다. 비동기 통신은 전송속도를 일치시키기 위해 전송 속도를 맞추는데 이를 BPS(Bit Per Sec)라 하고 1초 동안 전송하는 비트수를 의미한다. 직렬통신은 송신측과 수신측의 데이터 진행 경로의 구성 방법에 따라 반이중 방식(Half duplex)과 전이중 방식(Full duplex)으로 분류한다. 반이중 방식은 하나의 회로가 송수신을 겸하고 있고, 전이중 방식은 송신측과 수신측에서 각각 별도로 송수신을 행할 수 있도록 구성된 것을 말한다. 직렬통신의 대표적인 전기적 규격은 RS-232C로 전송거리가 짧고 저속인 경우에 사용하는 방법이다.

본 작품에서 직렬통신은 건반 번호를 PC의 하이퍼터미널에 모니터링 하고, MIDI 프로토콜을 이용해 PC에 음계을 전달하기 위해 사용하였다. ATmega164A는 두 개의 직렬통신을 지원하는데 USART0는 19600bps로 하이퍼터미널을 통한 직렬통신, USART1은 31.25Kbps로 MIDI규격을 이용한 직렬통신으로 이용하였다. 작품에서의 직렬통신은 비동기식 통신이며 전이중 방식으로 전기적 규격은 RS232C를 이용하였다. RS232C는 외부 노이즈 제거를 위해 높은 전압을 사용하므로 ±12V 전압을 필요로 하지만, 통신라인의 전압은 +5V와 0V를 사용하기 때문에 전기적 규격이 맞지 않아 이러한 차이를 맞춰주는 MAX232를 사용하였다.

2-2-2. MIDI

MIDI(Musical Instrument Digital Interface)는 컴퓨터 및 전자악기들 사이에 접속되어 음악 연주에 관련된 각종 정보들을 교신하는 하드웨어와 이를 제어하는 프로토콜의 국제적인 규격이다. MIDI 규격용 속도는 31.25Kbps이다. 본 작품에서는 MIDI규격을 이용해 피아노와 컴퓨터를 접속하여 피아노 연주 상황이 실시간으로 컴퓨터에 전달되어 악보로 저장하게 하였다. MIDI의 모든 통신은 마스터로부터 메시지 단위로 전송된다. <표 1>은 MIDI Channel Voice Message로, MIDI 메시지 규격을 나타낸다. 메시지 바이트 최상위 비트가 '1'이면 Status byte로서 메시지 시작을 의미한다. <그림 7>는 MIDI 규격을 이용해 데이터 전송을 코드로 구현하였다. 따라서 본 작품에서는 MIDI규격에 맞춰서 데이터를 직렬통신으로 PC로 보내 통신하였다.

기능	Status byte	2nd Data Byte	3rd Data Byte
	데이터 의미		
Note Off	1000 0000	0kkk kkkk	0vvv vvvv
	kkk kkkk: 음계 값 vvv vvvv: 소리 세기 값		
Note On	1001 0000	0kkk kkkk	0vvv vvvv
	kkk kkkk: 음계 값 vvv vvvv: 소리 세기 값		

<표 1> MIDI Channel Voice Message

```
;----------------------- MIDI 메시지
        ldi             R16,0b10010000          ; Note off message
        rcall       PUSH_MIDI_MSGQ
        ldi             R16,0x51                ; 음계값
        rcall       PUSH_MIDI_MSGQ
        clr             R16
        rcall       PUSH_MIDI_MSGQ
;----------------------- MIDI 메시지
        ldi             R16,0b10010000          ; Note on message
        rcall       PUSH_MIDI_MSGQ
        ldi             R16,0x51                ; 음계값
        rcall       PUSH_MIDI_MSGQ
        ldi             R16,0b01111111
        rcall       PUSH_MIDI_MSGQ
```

<그림 7> MIDI를 이용한 데이터 전송

2-2-3. 타이머/카운터

타이머/카운터는 CPU Clock을 이용하여서 일정한 주기에 맞춰 CPU에 인터럽트를 요구하는 것을 말한다. 타이머/카운터는 8비트와 16비트 두 종류가 있는데 본 작품에서는 타이머/카운터-1인 16비트를 사용했다. 타이머/카운터 카운트 할 클록신호가 주 카운터로 입력되면 카운터 값이 상승하며 입력해 준 비교 값 레지스터(OCR)의 값과 비교해 서로 동일하면 비교된 결과의 출력이 나타난다. 8비트 타이머/카운터와 달리 16비트 타이머/카운터에서는 OCR1A와 OCR1B의 두 개가 설치되어있다. 이때 AVR 프로세서의 주 클록이 매우 높아서 필요로 하는 만큼 적당한 주파수로 낮추기 위해 프리스케일러(Pre-Scaler)로 입력해 주파수를 낮춘 후 주 카운터로 입력된다. 따라서 타이머/카운터 회로를 사용하기 위해서는 프리스케일러 값, 비교 값 값을 설정해야한다. 또한 본 작품에서는 주기적으로 인터럽트를 발생하기 위하여 CTC(Clear Timer on Compare Match) 모드를 사용한다. CTC모드는 카운터가 증가하다가 OCR값과 일치하면 카운터를 다시 0으로 지워 처음부터 카운트 하도록 하는 것이다. 이때 카운터 값과 OCR 값이 일치 했을 때 인터럽트가 발생하게 된다. 단, CTC 모드 사용할 때는 OCR1A 하나만 사용할 수 있다.

본 작품에서 피아노 건반의 총 개수는 32개이고, 인터럽트가 한 번 발생할 때 건반 하나를 검출하도록 설계하였다. 건반 32개를 전부 검출하는데 걸리는 시간을 1ms로 정했을 때 1ms 동안 총 32번의 인터럽트가 발생해야한다. 결국 1s 동안 32000번의 인터럽트가 발생해서, 32KHz 마다 인터럽트가 발생하도록 설정하였다. <표 2>에서는 CPU Clock을 이용한 32KHz 계산 과정을 나타내었고 최종적으로 Pre-Scaler는 64분주, OCR은 3.9로 하였다. <그림 8>에서는 타이머/카운터 설정을 통해 32KHz 인터럽트를 구현한 코드이다.

CPU Clock	Pre-Scaler		목표주파수	OCR
8000000	1	8000000	32000	250
	8	1000000		31.25
	64	125000		3.9
	256	31250		0.97
	1024	7812.5		0.244

<표 2> CPU Clock을 이용한 32KHz 계산

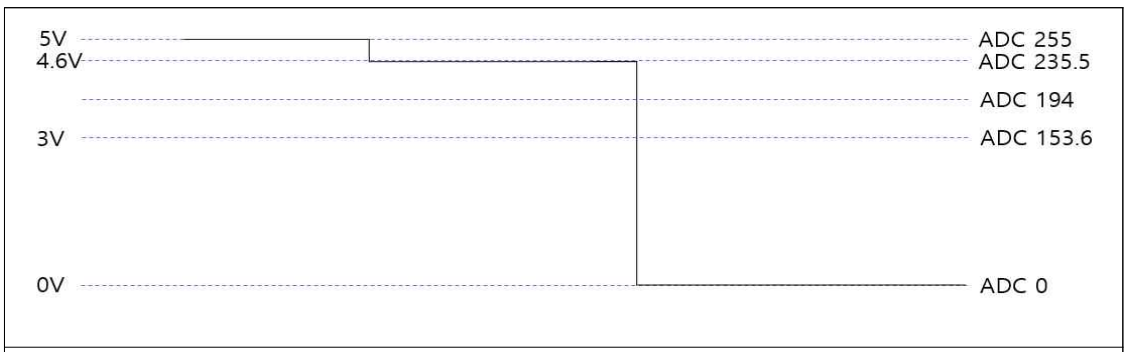
```
ldi     R16,0b00000000      ; TCCR1A
                            ;  7, 6 : COM1A - 사용 안함 00
                            ;  5, 4 : COM1B - 사용 안함 00
                            ;  3, 2 : 사용 안함 00
                            ;  1, 0 : WGM11, WGM10 - CTC mode 00
sts     TCCR1A,R16
ldi     R16,0b00001011      ; TCCR1B
                            ;  7, 6, 5 : 사용 안함 000
                            ;  4, 3 : WGM13, WGM12 - CTC mode 01
                            ;  2, 1, 0 : Prescaler - 011 (64분주) (8MHz MCU, 32kHz 목표)
sts     TCCR1B,R16

ldi     R16,HIGH(4)
sts     OCR1AH,R16
ldi     R16,LOW(4)
sts     OCR1AL,R16
ldi     R16,0b00000010      ; TIMSK
                            ;  1 : OCIE1A - OCR1A 인터럽트만 가동
sts     TIMSK1,R16
```

<그림 8> 타이머/카운터 설정을 통해 32KHz 인터럽트를 구현

2-2-4. ADC (Analog-to-Digital Converter)

아날로그/ 디지털 변환회로는 아날로그 신호가 입력되면 그에 해당하는 디지털 신호를 출력하는 것을 말한다. ATmega164A에는 ADC기능이 내장되어 있다. 아날로그 신호가 입력되면 아날로그 전압을 정해진 시간마다 측정해 측정된 전압을 정해진 디지털 전압에 해당하는 값으로 변환하는 것을 의미한다. 이때 정해진 시간마다 체크하는 것을 샘플링(sampling)이라 한다. 샘플링 간격이 너무 좁으면 데이터 량이 너무 많아져서 저장할 때 용량 문제가 발생하고 통신 속도가 저하하는 문제가 발생하기 때문에 적당한 간격으로 샘플링을 해야 한다. 샘플링 주기 이외에 아날로그의 전압을 구분하는 비트 수를 분해능이라 하며, 변환할 전압구간을 설정하는 것이 Vref (Voltage Reference)이다.

본 작품에서 변환할 전압구간 설정을 Vref을 기준전압 5V로 하였고, 8-비트 A/D 변환을 하였다. 다음 <그림 9>은 5V를 256 단계로 나누어 ADC로 변환한 것이다. 고스트 키에 걸리는 전압은 4.6V로 ADC 값으로는 약 235.5이다. 따라서 ADC값 194를 경계선으로 정해 194이상은 고스트 키로 처리하였다. <그림 10>은 이를 구현한 코드이다..



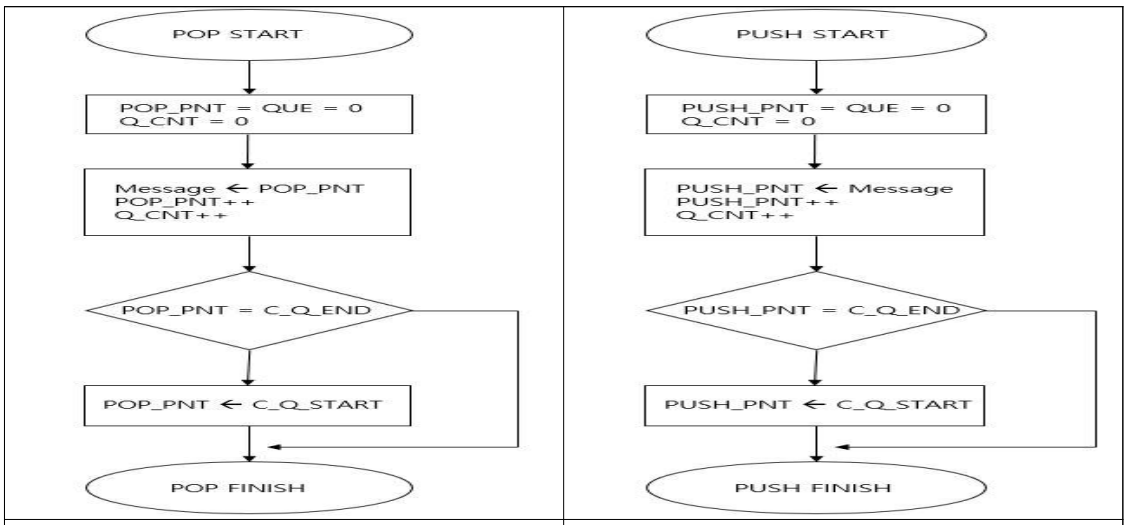
<그림 9> ADC

```
;=============================
; A/D 변환하기
;     약 235 ~ 236 근처면 고스트
;     약 194 이하이면 눌린 것
; 입력
;   R16 : 채널#
; 출력
;   R16 = 00 - 눌렸음
;   R16 = FF - 안눌렸음
;=============================
SUB_ADC:

        ori             R16,0b01100000          ; REFS1,0 = 01 - Vref = AVCC
                                                ; ADLAR = 1 (상위 8비트만 사용)
        sts             ADMUX,R16

        ldi             R16,0b11010000          ; ADCSRA 설정
                                                ;   b7 : ADEN = 1(ADC 가동)
                                                ;   b6 : ADSC = 1(ADC 시작)
                                                ;   b5 : Auto = 0
                                                ;   b4,3 : ADC INT = 00
                                                ;   b2,1,0 : Prescaler = 000(최대속도)
        sts             ADCSRA,R16
SUB_ADC_0:
        lds             R16,ADCSRA
        sbrs    R16,4
        rjmp    SUB_ADC_0

        lds             R16,ADCL
        lds             R16,ADCH

        cpi             R16,194                 ;194 경계값 비교
;       cpi             R16,250
        brcc    SUB_ADC_1
        clr             R16
        ret
SUB_ADC_1:
        ser             R16
   ret
```

<그림 10> ADC를 이용한 고스트 키 처리

2-2-4. 메시지 큐 (Message Queue)

피아노에서는 여러 개의 건반이 눌렸을 때 순차적으로 데이터를 처리해야하는 구조가 필요하다. 시간차에 따라 먼저 눌린 건반과 나중에 눌린 건반이 따로따로 순서대로 처리 되어야 하며, 동시에 눌린 건반들은 한 번에 처리해야한다. 본 작품에서는 메시지 큐를 구현하여 이러한 피아노의 동작을 해결하였다. 큐는 한쪽 끝으로 데이터를 입력받고, 반대쪽에서는 데이터를 출력하여 데이터를 순차적으로 처리하는 구조이다. 아래 <그림 11>에서는 메시지 큐에 데이터를 입력받고 출력하는 PUSH, POP알고리즘을 구현한 것이다.



POP 알고리즘 | PUSH 알고리즘

<그림 11>

2-2-3. 피아노 건반 상태에 따른 검출

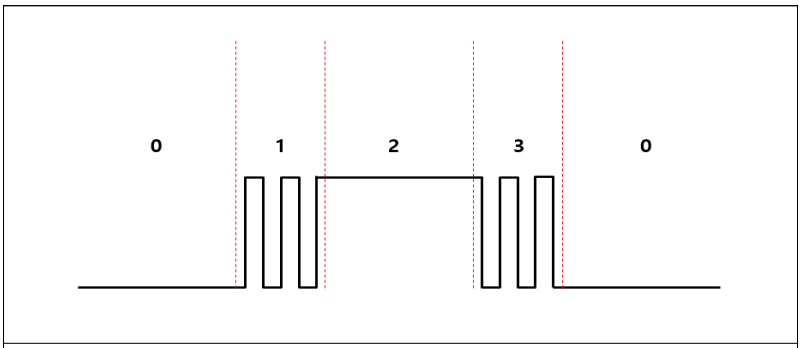
피아노 건반은 각각이 주기적으로 한 번씩 점검되며 이때 키가 어떤 상태에 있는지 상태정보를 유지하기 위해 스위치 상태마다 플래그 변수를 할당한다. 아래 <그림 12>은 피아노 건반이 눌렸을 때 건반 상태 그림으로서 각 상태마다 플래그를 선언하였음을 볼 수 있다.

상태-0 피아노 건반이 눌리지 않은 상태

상태-1 피아노 건반이 눌리고 발생하는 채터링을 제거하는 상태

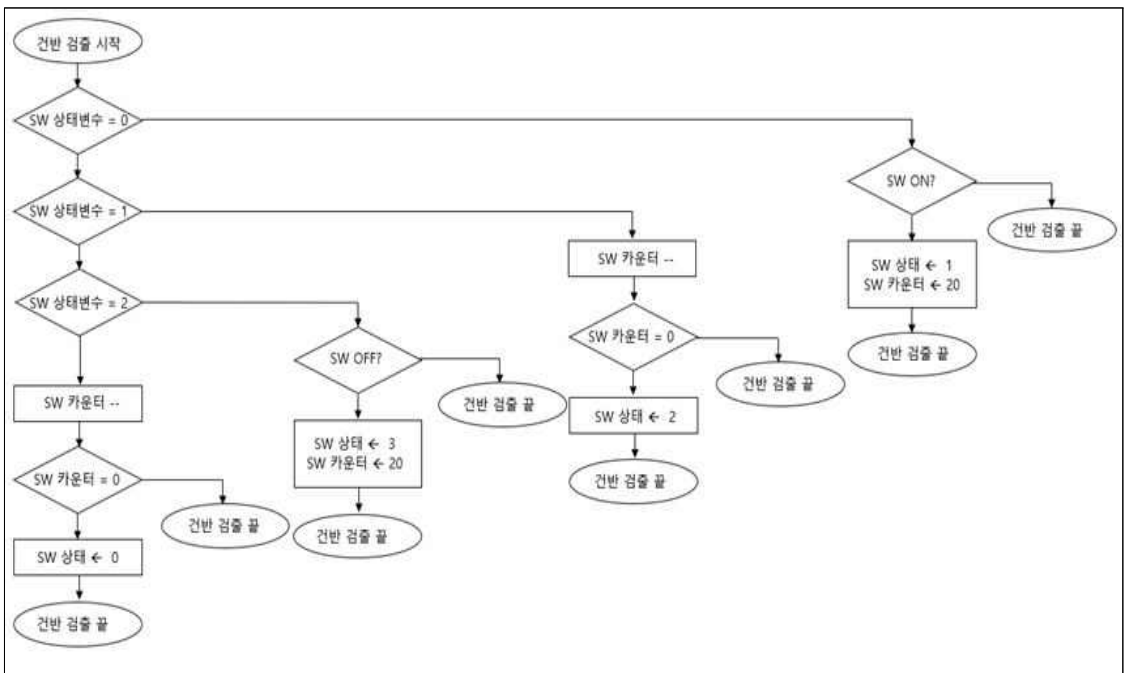
상태-2 피아노 건반이 눌려있는 상태

상태-3 피아노 건반이 오프된 후 발생하는 채터링을 제거하는 상태



<그림 12> 스위치 상태에 따른 플래그 선언

한 개의 건반이 눌렸을 때 다른 건반이 인식되지 않는다면 피아노 연주가 불가능해 진다. 따라서 플래그를 설정하여 총 32개 건반을 따로따로 검출하게 되면 동시에 여러 가지 키를 눌렀을 때 건반별로 피아노 음을 처리 할 수 있게 된다. 아래 <그림 13>은 스위치 상태에 따른 코드 알고리즘이다.

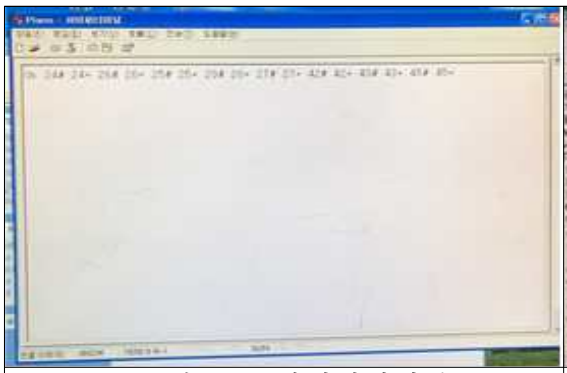


<그림 13> 스위치 상태에 따른 코드 알고리즘

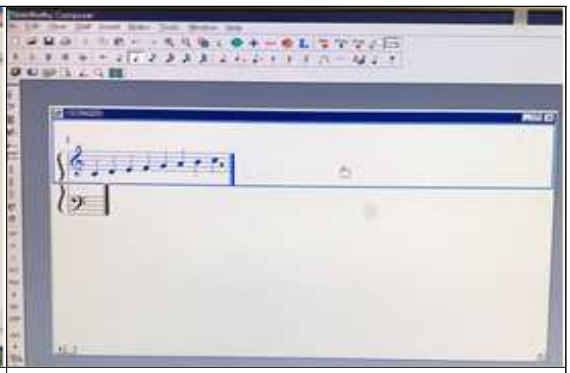
제3장 결론 및 작품제작 후기

작품을 제작하면서 다이나믹 스캐닝을 이용해 프로세서로 피아노 건반을 검출하였다. <그림 14>에서는 검출된 건반을 직렬통신을 이용해 하이퍼터미널에 데이터를 출력하도록 하고, <그림 15>에서는 MIDI 프로토콜을 이용해서 피아노 건반의 음계를 전송해서 피아노 악보를 출력하도록 나타냈다. 건반 3개 이상을 눌렀을 때 발생하는 고스트 키를 ADC를 이용해 해결하였고, 건반 상태를 플래그로 피아노 32개의 건반을 각각 처리하여 피아노 연주 시 화음을 넣을 수 있도록 제작하였다. 하나의 건반이 눌렸을 때 다른 건반도 인식될 수 있도록 구현 하였다. <그림 16>은 작품 완성사진이다.

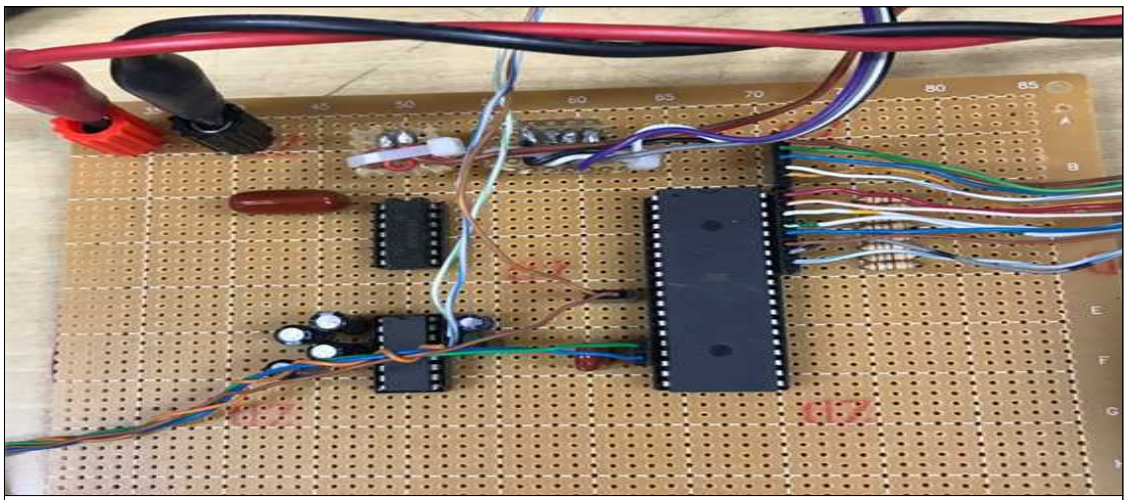
현재 본 작품에서는 전자 피아노를 건반 소리 이외에 다른 악기 소리도 낼 수 있는 기능과 LED를 이용해 피아노 연주시 건반별로 LED가 작동하는 기능을 추가하는 것이 추후 목표이다.



<그림 14> 하이퍼터미널



<그림 15> MIDI



<그림 16> 최종 작품 사진

참고문헌

[1] 이현창, “컴퓨터 주변회로”, 상학당
[2] 이현창, 유창근 “마이크로프로세서 구조 및 실습”, 남두도서