신디사이저와 IoT

(최종 보고서)

하프시코드
이름 : 조용래
학번 : 201224540
분과 : 컴퓨터 시스템 1
지도교수 : 김호원 교수님

1. 과제 목표 및 선정 이유

컴퓨터 공학적 기술과 전자 공작 기술 지식들을 활용하여 아두이노로 만든 통신 기능이 들어 있는 전자 악기(신디사이저)를 만든다.

중간 보고서때는 선정 이유를 거창하게 예술가들 운운하며 피지컬 컴퓨팅을 명목으로 잡았지만 다시 돌이켜보면 신디사이저 제작은 지금까지 배웠던 컴퓨터 공학적 지식이 잘 응용될 수 있는 영역이라는 판단이 들어서였다. 컴퓨터는 뿌리부터가 수학이고 음악 역시 마찬가지이다. 피아노 건반만 봐도 하나하나의 건반이 모두 특정 주파수의 음을 내는 것이고 음들의 지속 시간, 악보의 표기법 역시 수학의 형식적 요소를 빼다 닮은 것이다. 컴퓨터는 수학적 지식을 표현하는데 발군의 능력을 자랑한다. 수학적으로 표현될 수 있는 음악 역시 컴퓨터로 완전하게 표현될 수 있다. 그러한 표현 방법 중 하나인 미디 프로토콜을 이 프로젝트에서는 중요점으로 삼는다.

그렇다고 비중이 음악 쪽에서만 치중될 수는 없는 것은 아두이노로 하드웨어적 입출력을 구현하기 위해서는 각종 전자 공작 지식들이 많이 필요하다. 이런 지식들은 회로 이론의 어려운 이론들이 아니라 간단한 옴의 법칙이나, 공구 도구들 활용, 배선, 납땜, 각종 재료들의 활용 능력 등이 요구된다.

즉, 4년 간 컴퓨터 공학을 전공하면서 대개 스크린 속에서 비춰진 텍스트와 정보 조작에만 시간을 쏟았다면, 임베디드와 피지컬 컴퓨팅을 접하면서 모니터 밖에서 이뤄지는 사람과 컴퓨터의 감각적이고 적극적인 상호작용에 졸업 과제의 초점을 맞추었다.

2. 설계 상 변경 및 제약 사항

수정 사항 #1

이 프로젝트에서 소리를 만들어 내는 미디 보드의 제약에 대해 먼저 설명을 해야겠다. 미디 보드는 일반 미디 프로토콜(General MIDI)를 지원하지만 그 중에 일부만 지원한다. 예를 들어, 일반 미디는 포르타멘토나 트레몰로와 같은 음 변조 테크닉 명령을 명시해놓았지만 스파크 펀에서 제작된 미디 보드는 이러한 명령들을 지원하지 않는다. 이유는 미디 보드에 사용되는 칩(vs1053)이 미디 명령의 일부만 지원하기 때문인데 자세한 내용은 이후에 다시 다루겠다.

수정 사항 #2

그리고 중간보고서 발표 자료를 다시 보니 '음악을 전송'한다는 개념이 마치 바이너리 음악 파일을 주고 받고 재생한다는 늬앙스가 있는 것 같아 여기서 정정한다. 음악은 노트들의 나열이고 노트는 수치로 표현된다. 음악의 모든 것은 숫자와 형식으로 표현될 수 있고 전자 악기의 공통 프로토콜인 미디 언어로 재표현될 수 있다. 여기서는 미디가 음악을 어떻게 다루는지 이해하고 이를 바탕으로 음악을 노트들의 나열로 다루었다. (노트는 1~127사이의 정수값이다.) 자료 구조도 이에 맞게 재설계 되었다.

수정 사항 #3

중간보고서에는 입출력 보드(아두이노 메가)와 우노 보드와의 통신을 SPI라고 했는데 I2C 통신으로 수정한다. SPI는 SD카드와의 통신을 하기 위해 주로 쓰인다. SPI가 그래서 속도가 빠른 것이다. 메가에서는 SD카드를 음악 파일을 저장하고 읽거나 길다란 문자열들(악기 목록)을 읽기 위한 수단으로 쓰인다.

수정 사항 #4

중간 보고서에는 작은 스피커 사진을 올려 놨었는데 진행해본 결과 일반 스피커나 헤드폰만 있어도 충분하다. 따라서 필요 없어서 제거하였다.

수정 사항 #5

마지막으로 서버쪽에서의 통신은 우노나 클라이언트나 둘 다 TCP 통신으로 되어 있는데 좀 더 정확하게 말하면 우노 쪽과는 TCP 소켓 방식으로 통신을 하고 클라이언트 쪽으로는 웹 소켓 방식으로 통신한다. 클라이언트는 웹 페이지를 보여주는 브라우저를 대상으로 한다. (안드로이드/iOS 앱이 아니다.)

참고 사항

중간 보고서때는 미디 프로토콜에 대한 설명에 많은 지면을 할애하였다. 물론 전체적인 구조를 잘 설명한 것은 아니지만 꼭 필요한 기능이 있고 그렇지 않은 기능이 있다. 미디 메시지의 이해는 매우 중요하지만 생략하겠다.

3. 하드웨어 구성

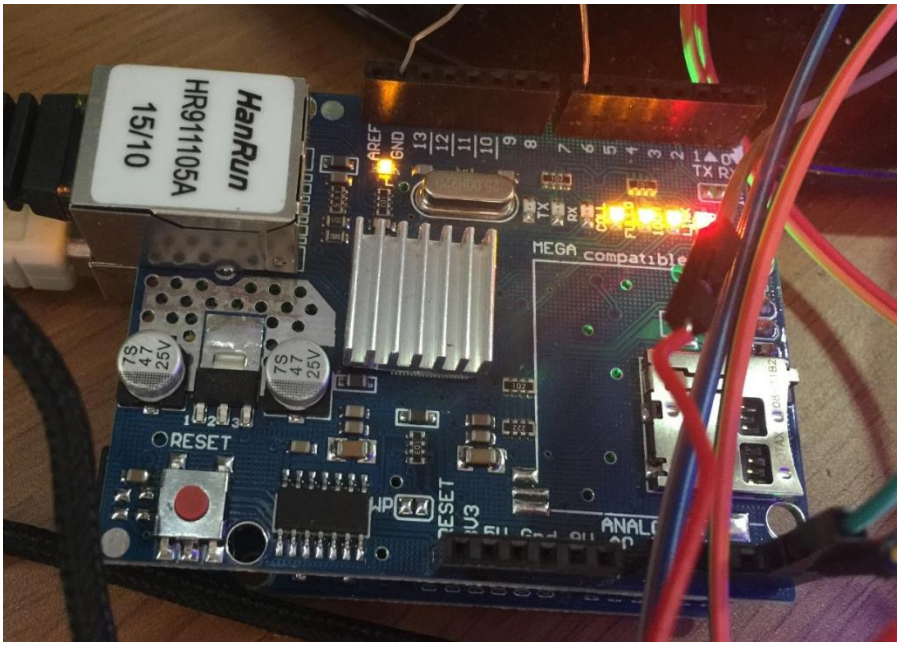
졸업 과제의 대부분의 시간이 사실 하드웨어들을 짜맞추고 테스트 하고 갈아 끼우고 하는 곳에 많이 투자 되었다. 그도 그럴것이 하드웨어는 4년 간 키보드만 두드리다가 이쪽 분야로 넘어와서 그런지 다루는데 능숙하지 않았다. 전선 피복하는 것도 첨엔 니퍼로 무식하게 힘만 쓰다 날려 버린 전선 조각이 한 두개가 아니다. 하지만 얼마 후에 와이어 스트리퍼와 전선 규격에 대한 이해가 선행된 이후에는 짧고 빠른 시간에 원하는 길이의 전선을 피복할 수 있게 되었고, 배선이나 납땜 같은 다른 익숙치 못 했던 기술들 또한 얼마간의 노하우가 생긴 후에는 꽤나 안정적으로 원하는 결과를 얻어낼 수 있었다.

하드웨어 구성은 구축 대상의 물리적 토대를 결정한다. 이후의 소프트웨어는 이런 토대위에 구축되기 때문에 하드웨어 조각들의 모음은 중요하다. 하다보니 이것저것 많이 쓰게 되었는데 하나씩 차례대로 각 기능과 쓰게 된 이유를 간략하게 사진과 함께 설명하겠다.

우노

이더넷 쉴드가 부착된 아두이노 우노 보드는 악기(메가)와 서버와의 통신을 위해 사용된다. 중간

매개자 역할을 하는 셈인데, 서버 쪽에서 받은 JSON 형식의 문자열을 오픈 소스 ArduinoJSON을 사용하여 파싱하여 해당 값들을 I2C 방식으로 연결된 메가에게 넘겨 준다. 다음 그림은 아두이노 우노 보드이다. 과제 수행 중에 굳이 통신 역할을 메가 쪽으로 몰아서 해도 되는데 왜 굳이 우노와 메가로 분리해서 구현하려고 하는가에 대한 의문이 있었다. 그때 당시에는 통신량 자체가 적



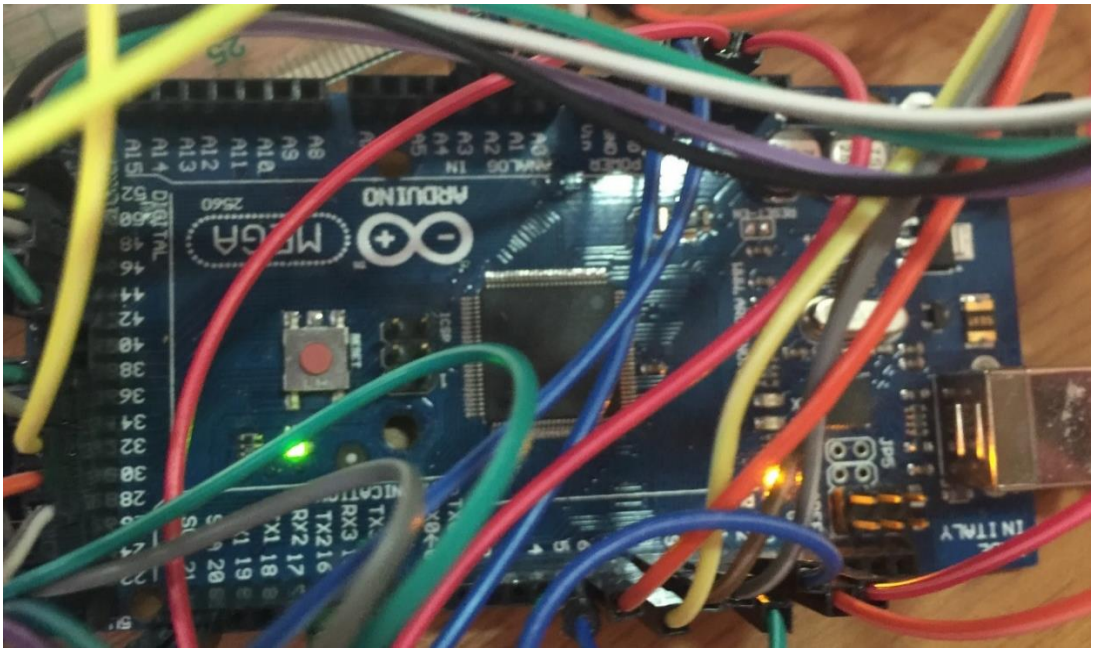
<그림 1> 이더넷 쉴드가 부착된 아두이노 우노

었으니까 별로 문제가 되지 않았지만 JSON을 파싱하는 것이나 이더넷 라이브러리 활용에 들어가는 메모리 용량, 그리고 이후의 추가적으로 통신 파트에 들어갈 기능과 같은 확장성을 고려해보니 성격이 서로 다른 역할을 하는 대상은 물리적으로도 그렇게 분리 시켜 놓는 게 좋다는 설계상의 판단이 들었다. 레이어를 한 층 더 쌓는 것은 복잡성을 증대시키지만 그 만큼 구조는 견고해지기 마련이다. (여기서 확장성을 언급한 이유는 졸업 과제 이후에도 꾸준 발전 시킬 가능성이 있음을 함의한다.)

메가

신디사이저를 한 번이라도 본 사람이라면 키보드 건반 위로 수 많은 노브들과 슬라이더, 버튼들과 LCD 등과 같이 사용자가 조작할 수 있는 수단이 다양하다는 것을 알 것이다. 이 말은 아두이노로 신디사이저를 제작하기 위해서는 그 만큼 많은 입출력 핀이 필요하다는 뜻이다. 우노는 핀 수가 적지만 메가 보드는 많다. 물론 핀 수 확장을 위한 멀티플렉서 모듈(4 input to 16 output)이 있기는 하지만 메가의 핀 수라면 별로 문제가 되지 않고 먹스를 사용할 경우 귀찮은 배선을 추가로 해야한다는 단점이 있다. 메가는 그리고 입출력 핀 뿐만 아니라 메모리 용량도 우노에 비해 많기 때문에 좀 더 자유로운 프로그래밍이 가능하다. 하지만 과도한 동적 할당이나 연속 메모리 할당 메커니즘을 사용하지 않는 것이 좋다. 이 부분에서 잠시 할 말이 있는데, 데탑에서와 임베디드 보드와의 결정적 차이점은 OS존재의 유무이다. 동적 할당시 임베디드에서는 OS가 관리를 못

해주기 때문에 프래그먼테이션 현상이 종종 일어나고 이는 보드를 일순간에 마비시켜 버린다. 연속 메모리 할당 문제는 긴 문자열을 한 배열에 집어 넣으려고 할 때 다른 영역에 있는 자료를 침범할 때 나타나는데, 워낙 메모리 자체가 데탑에 비해 작다 보니 이러한 문제가 생긴 것이다. 혹시나 Flash Memory 용량(256KB)을 보고 충분하다고 생각할 수도 있으나 아두이노는 SRAM(8KB)



<그림 2> 아두이노 메가

와 EEPROM(4KB)에 자료가 저장되고 이들의 용량은 하나의 변수가 과도하게 클 경우가 문제가 생길 수 있다. (참고로 필자는 100가지 이상이나 되는 악기 목록 전부 메모리에 로딩하려다가 에러 먹어서 대체 왜이러나 싶었지만 이 문제가 메모리 연속 할당이 문제라고 생각하였고 이를 SD 카드로 옮김으로써 해결하였다.) 다음은 메가 보드 사진이다. 중간 보고서에 있는 사진과는 다르게 선들이 많이 보인다.

미디 보드

메가 보드가 악기에서의 두뇌 역할을 한다면 미디 보드는 손발 역할일 것이다. 악기에서 가장 중요한 소리를 내는 알고리듬을 내장한 칩을 가지고 있다. 서두에 일반 미디 명령 중 일부만 지원한다고 했는데 다음이 그 목록이다. 신디사이저에서 자주 쓰이는 애프터터치나 포르타멘토 트레몰로와 같은 고급 명령어들은 지원하지 않는다. 그나마 지원하는 명령 중에 기본적으로는 노트 온/오프, 채널 볼륨, 음색 뱅크 선택이 있고 컨트롤 체인지 명령어로는 리버브나 팬 포트가 있다. 서스테누토나 홀드는 피아노 페달을 의미하는데 이들 값들을 변화시켜 보면서 테스트해본 결과 별 차이를 못 느껴서 사용하지는 않았다.

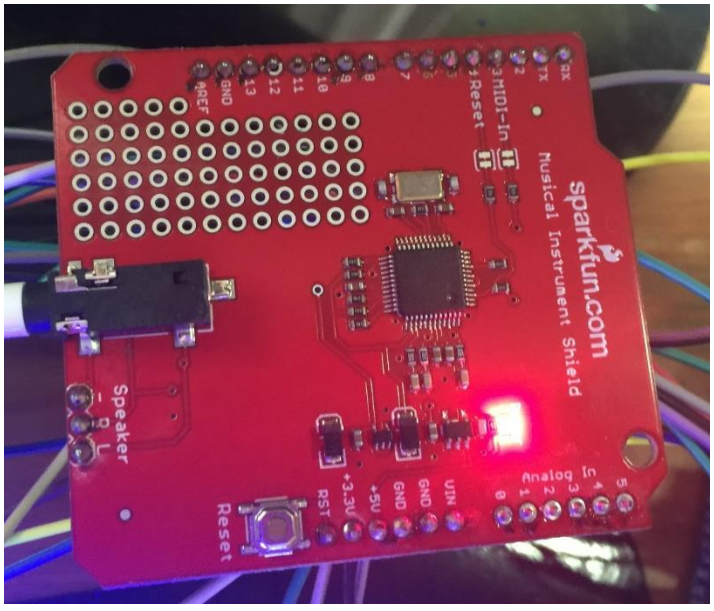
미디 보드에 내장된 칩은 VS1053b 이지만 스파크 펀에서 제조한 Musical Instrument Shield는 명세된 이 칩의 기능 중 미디 부분 일부만 사용 가능하도록 만들어 놓았다. 여기서 현실적 제약 사

항이 나온다. 처음에는 미디 명령어 대다수를 지원하며 음 자체를 자유자재로 변조해보면서 주파수 변화의 매력을 제공하려고 했지만 안타깝게도 하드웨어 지원의 한계 때문에 제한된 기능들만 제공하게 된다는 점이다. (참고로 이런 문제를 8월 초에 깨닫고 해외 사이트에서 미디만을 전용으로 다루는 CPU를 구매했었지만, 사용 방법이 생소하였고 시간 상의 제약 때문에 스킵하였다.)

Supported MIDI messages:

- meta: 0x51 : set tempo
- other meta: MidiMeta() called
- device control: 0x01 : master volume
- channel message: 0x80 note off, 0x90 note on, 0xc0 program, 0xe0 pitch wheel
- channel message 0xb0: parameter
  - 0x00: bank select (0 is default, 0x78 and 0x7f is drums, 0x79 melodic)
  - 0x06: RPN MSB: 0 = bend range, 2 = coarse tune
  - 0x07: channel volume
  - 0x0a: pan control
  - 0x0b: expression (changes volume)
  - 0x0c: effect control 1 (sets global reverb decay)
  - 0x26: RPN LSB: 0 = bend range
  - 0x40: hold1
  - 0x42: sustenuto
  - 0x5b effects level (channel reverb level)
  - 0x62,0x63,0x64,0x65: NRPN and RPN selects
  - 0x78: all sound off
  - 0x79: reset all controllers
  - 0x7b, 0x7c, 0x7d: all notes off

<그림 3> VS1053b 지원하는 미디 메시지

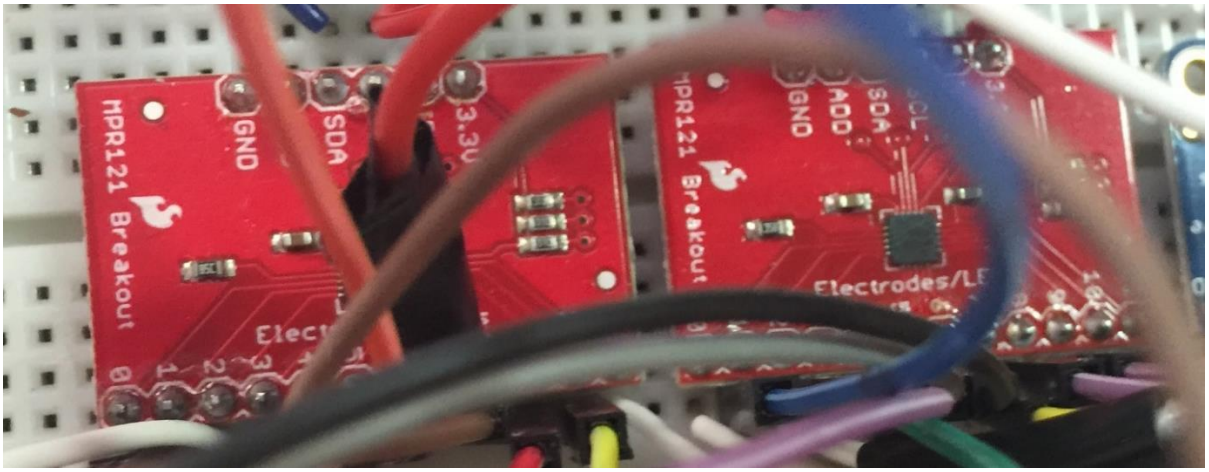


<그림 4> 미디 보드

정전식 터치 센서(MPR121)

미디 보드도 이 센서가 없으면 키보드를 만들어 내지 못할 것이다. I2C 프로토콜로 연결되는 이

정전식 터치 센서는 총 12개의 정전식 접점을 가진다. 12라는 숫자는 피아노의 한 옥타브와 일치한다. 그래서 이 센서를 처음 사용했을 때 딱 피아노 건반으로 연결하면 좋겠다는 생각이 들었다. I2C에서의 주소는 0x5A, 0x5B, 0x5C, 0x5D를 가질 수 있다. 이는 해당 센서의 ADDR핀을 재설정해줌으로써 가능하다. 여기서는 스파크 펀에서 나온 보드를 사용하였는데 ADDR핀은 처음에 GND에 연결되어 있는데 뒷면에 보면 가느다란 선이 있는데 이를 칼로 그어 GND 쪽으로의 선을 잘라준 다음에 다른 핀으로 연결해야 I2C에서의 주소가 제대로 인식된다. 정전기 현상을 입력으로



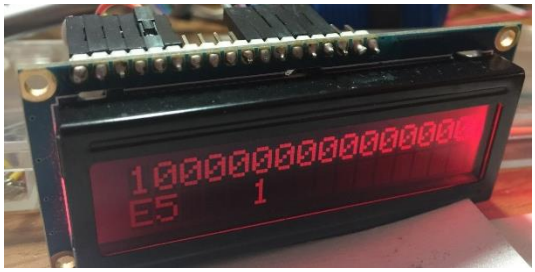
<그림 5> MPR121 (정전식 터치 센서)

Pin No.	Pin Name	Description
1	$\overline{IRQ}$	Active Low Open-drain Interrupt Output
2	SCL	I^2C Serial Clock
3	SDA	I^2C Serial Data
4	ADDR	I²C Slave Address Pin Selects. Connect to VSS, VDD, SDA, SCL to choose address 0x5A, 0x5B, 0x5C, 0x5D respectively.

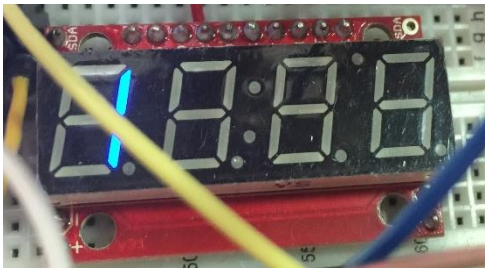
받아 들이기 때문에 건반은 금속판이 적당하다. 피 아노 건반 구축은 이후의 피아노 건반 부분에서 다룬다.

LCD, 7segment

LCD나 7segment는 악기에서 디스플레이 역할이다. LCD에는 시스템 메시지나 건반 위치, 현재 선택된 마디나 방금 누른 건반 키값, 현재 옥타브 따위를 표현한다. 세그먼트는 LCD 를 보조해주는 역할을 한다. 주로 방금 변화된 수치 값을 찍어낸다.



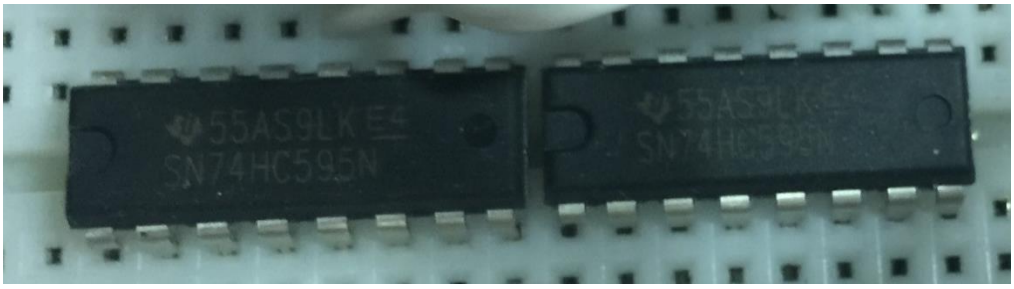
<그림 7> LCD



<그림 8> 7세그먼트

쉬프트 레지스터

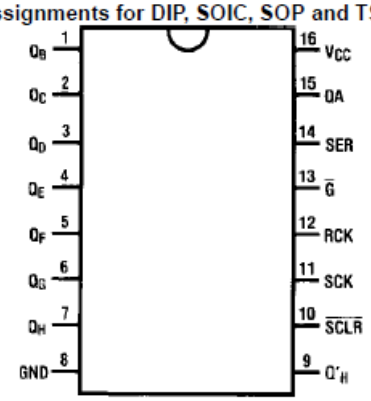
아두이노가 아무리 핀이 많다지만 16 * 5 = 80개나 되는 LED를 꽂을 핀은 없다. 다수의 입력 핀이 필요한 경우 앞서 언급한 먹스가 있지만 출력 핀에는 디코더나 쉬프트 레지스터가 있다. 디코더는 배타적 출력을 내기 때문에 동시에 여러 LED를 점등해야할 상황일 때는 부적합하다. 쉬프트 레지스터는 1바이트 짜리 시리얼 입력을 받는데 이때 1의 위치에 따라 해당 핀의 출력에 전류를 흘려 LED를 킨다.



<그림 9> 쉬프트 레지스터(74HC595) 아두이노 포럼에서 자주 볼 수 있는 IC이다.

Conne

Pin Assignments for DIP, SOIC, SOP and TSSOP



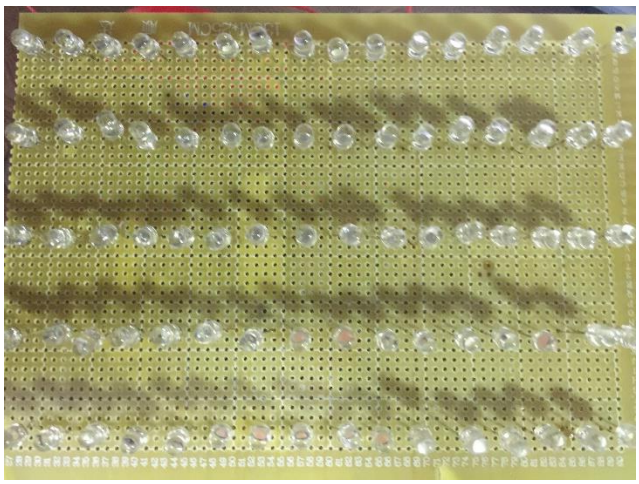
Top View

RCK	SCK	$\overline{SCLR}$	$\overline{G}$	Function
X	X	X	H	Q_A thru Q_H = 3-STATE
X	X	L	L	Shift Register cleared $Q_H = 0$
X	↑	H	L	Shift Register clocked $Q_N = Q_{n-1}$, Q_0 = SER
↑	X	H	L	Contents of Shift Register transferred to output latches

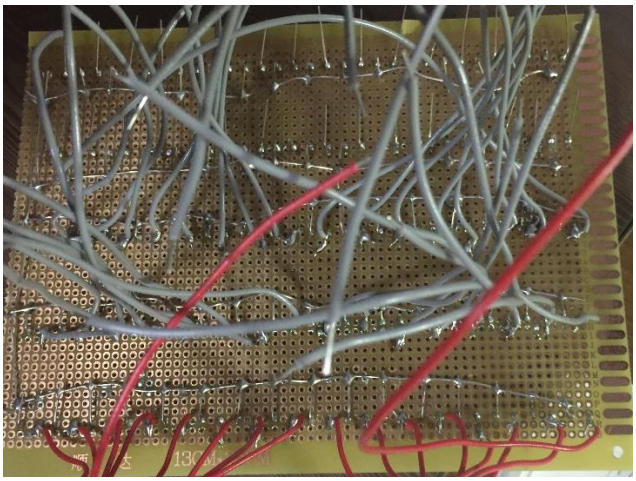
<그림 10> 쉬프트 레지스터 핀 할당 – 래치 핀이 하이로 올라 가는 시점에(에지 트리거) 쉬프트 레지스터 값이 출력 래치로 이동함을 알 수 있음 -> 이를 바탕으로 코딩해야 함.

LED 패널

LED 패널은 80개의 LED가 납땜이 각 라인에 16개씩 총 5개의 라인으로 납땜이 되어 있는 PCB 보드이다. 16개라는 숫자는 한 마디를 16분음표 16개로 채우겠다는 의미에서 그렇다. 5개는 각기 다른 채널의 마디를 의미하는데 해당 마디에서 활성화된 노트를 뜻한다. 노트 하나씩은 일정한 시간 간격 뒤에 켜지거나 꺼지는데 이때 타이머 콜백 함수가 이용된다. 이 함수가 호출 되는 주기에 맞춰 음을 내거나 끈다. 이에 맞춰 LED도 켜지거나 꺼지는 식으로 보여진다.



<그림 11> LED 패널(앞)

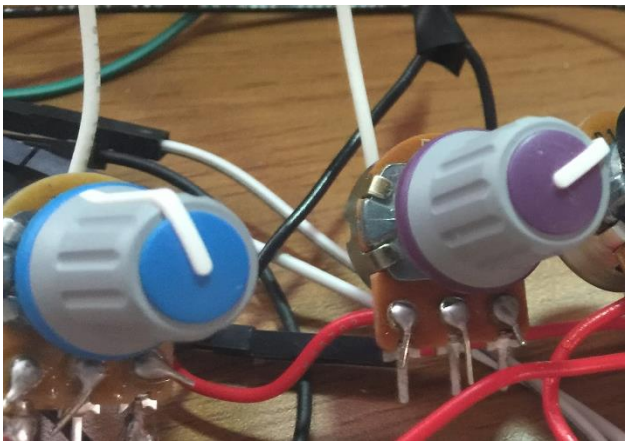


<그림 12> LED 패널(뒤) : 아직 선 연결이 안 된 부분이 있음

스위치들

가변 저항

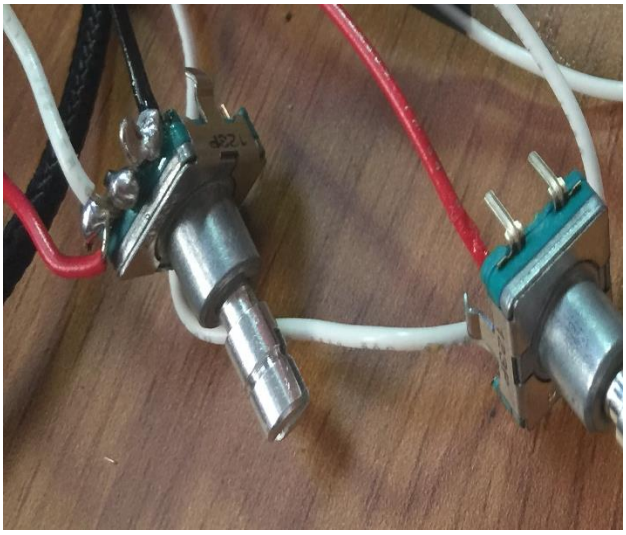
볼륨, 템포, 리버브, 팬포트와 같이 값 변화에 민감하게 반응하지 않는 요소들로 선택한다. 왜냐하면 가변 저항은 전류를 흘려나가는 양을 저항 값을 조절함으로써 입력이 들어가는데 종종 전류가 불안정할 때가 있기 때문에 변화를 주지 않아도 값이 조금씩 변할 때가 있기 때문이다.



<그림 13> 가변 저항

로타리 엔코더 스위치

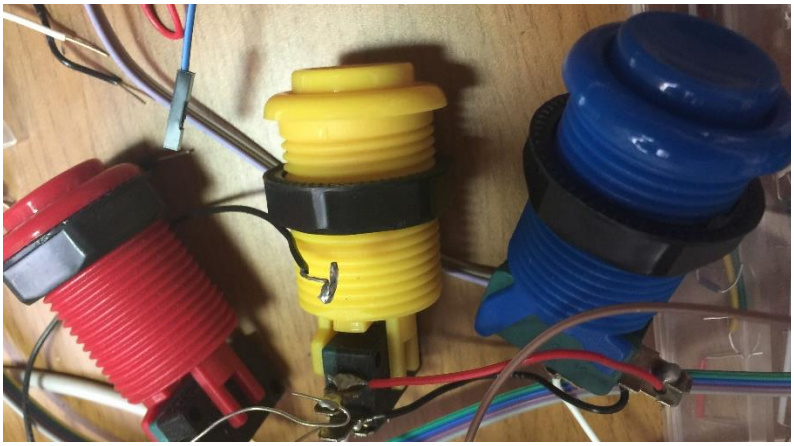
가변 저항의 문제를 해결하게 위해 로타리 엔코더 스위치가 필요하다. 노트 위치 조절이나, 음색 변경, 채널 변경에 사용된다. 2개의 핀을 사용하는 이 스위치는 값 변화가 일어났느냐의 여부(토글), 방향(왼쪽 또는 오른쪽)의 정보를 0, 1로 받아 소프트웨어적으로 변수의 값을 변화시키는 것이다. 아두이노에서 엔코더 스위치를 제어하는 오픈 소스가 있어서 약간 변형한 후 사용하였다.(원본은 인터럽트 호출 시에 사용되는 것인데 반면에 여기서는 토글이 일어났을 때 사용하는 것으로 변경)



<그림 14> 로타리 엔코더 스위치

게임 스위치

오락실 게임기에 쓰이는 스위치인데 재밌어 보여서 구매했다. 한 마디를 재생하거나 정지 또는 현재 위치의 노트를 제거하는데 사용된다.



<그림 15> 게임 스위치

택트 스위치

범용적인 스위치로 많이 쓰이는 스위치다. 아직 음악 소프트웨어 설계에 대해 자세히 설명하지 않아서 이전 설명에 잘 이해 안가는 부분이 있을 것 같다. 이 스위치는 현재 선택된 마디에서의 다음 마디, 이전 마디, 마디 추가, 마디 삭제 등의 마디 연산이나 현재 음에서의 반음 쉬프트 기

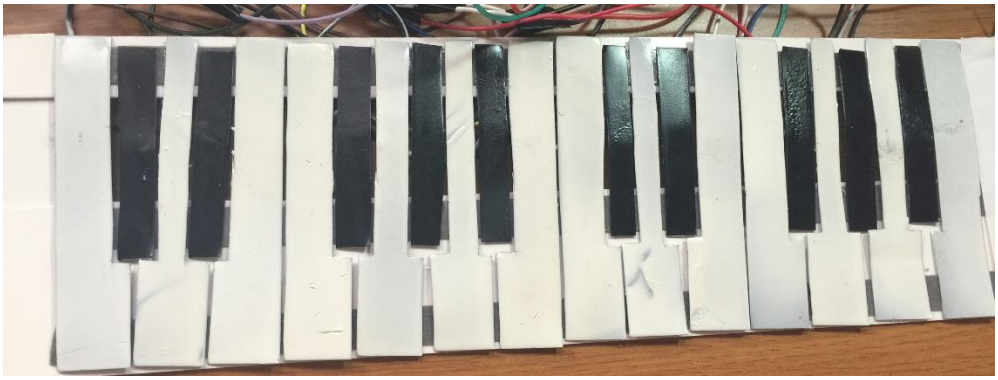
능을 제공한다. 반음 조절은 악전(musical notation)으로는 샾이나 플랫으로 표현한다.

<그림 16> 택트 스위치들

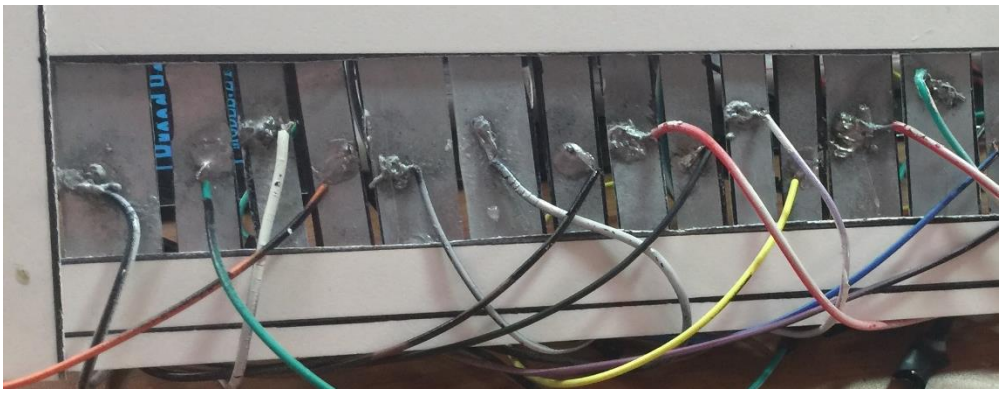
피아노 건반

하드웨어 구성 중에 가장 많이 갈아 치웠던 부분이 이것이다. 처음에는 금속판이면 전선을 강력 테이프로 접착해서 이어 붙이면 되지 않을까라고 생각해서 그렇게 하다가 건반이 이리저리 이동하는 과정에서 접촉 불량이 계속 생겨서 전도성 테이프로 바꾸었다. 처음에는 잘 되더니 건반을 들었다 놨다 몇 번 반복하니 한 두 군데에서 접촉 불량이 계속 생기게 되었다. 결국에는 납땜으로 강하게 결합시키는 것이 필요하다고 생각해서 기존 접촉들을 다 떼어내고 다시 붙이게 되었다. 이때 사용된 판은 아연판이다. 알루미늄판, 철판, 구리판을 써봤지만 알루미늄은 납땜 페이스트를 발라도 납 접착이 잘 안되는 반편, 철판은 강도가 너무 쎄서 자르기가 힘들고, 구리 판은 색상이 너무 틔어서 피아노 건반 색상과는 잘 안 맞는다고 생각하였고 결국엔 아연판으로 선택하였다. (구매한 판은 보통 초등생 과학 실험으로 제작된 금속 판들이어서 얇음) 다음 그림은 피아노 건반의 위, 아래에서 찍은 사진이다.

정전식 터치 센서를 언급할 때 나왔던 12개의 접점이 피아노 한 옥타브에 정확히 맞아 떨어지는 것을 볼 수 있다. 색을 보면 하얀색과 검은색이 있는데 이는 스프레이로 색을 입힌 것이다.



<그림 17> 피아노 건반(위)



<그림 18> 피아노 건반(아래)

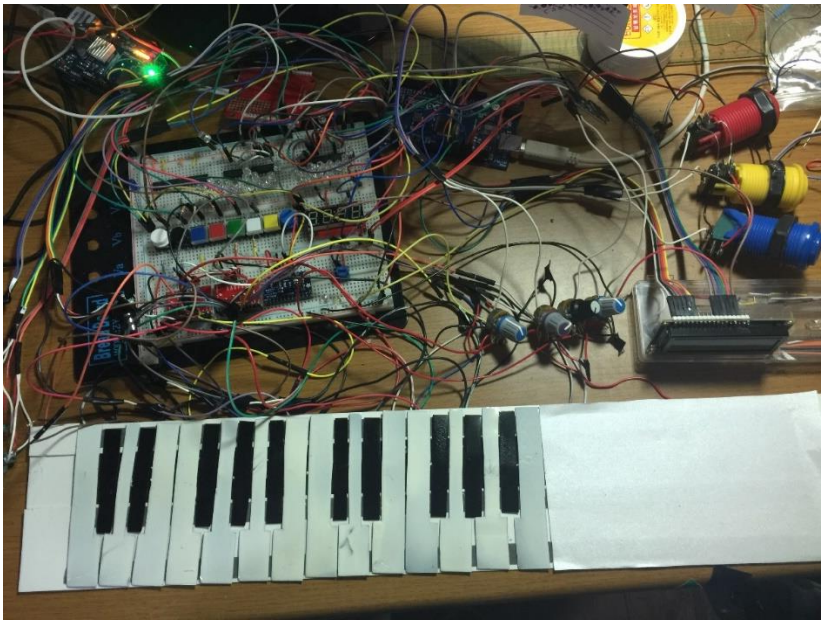
<그림 19> 아연판

<그림 20> 몬태나 스프레이(검/흰) : 처음엔 아연판을 흰 건반, 구리를 검은 건반으로 쓰려다가 스프레이를 사용하여 색상 효과를 나타냈다.

<그림 21> 스프레이 뿌리고 난 직후의 아연 건반들

이것으로 하드웨어 구성에 대한 설명을 모두 마쳤다. 처음부터 이렇게 조립해보라고 정해준 메뉴얼이 없었다. 부품 하나하나가 시행착오의 대상이다.

~~그럼 지금까지 설명한 모든 부품들을 하나로 묶어주는 본체가 필요한데 아직 이를 제작하지는 않았다. 왜냐하면 소프트웨어 작업이 마무리가 덜 되어있기 때문이다.~~ 아래 사진은 브레드 보드 위에 조립된 하드웨어들의 총집합이다.



<그림 22> 하드웨어 총 집합(조립 전)

<그림 23> 신디사이저(조립 후)

4. 소프트웨어 구성

소프트웨어 구성은 아두이노 메가에 들어나느 음악 소프트웨어, 우노에 들어가는 통신 소프트웨어, 서버와 웹 클라이언트 쪽 소프트웨어가 있다. 먼저 음악을 숫자로 표현하는 방법을 살펴 보는 것이 좋을 것이다.

악기

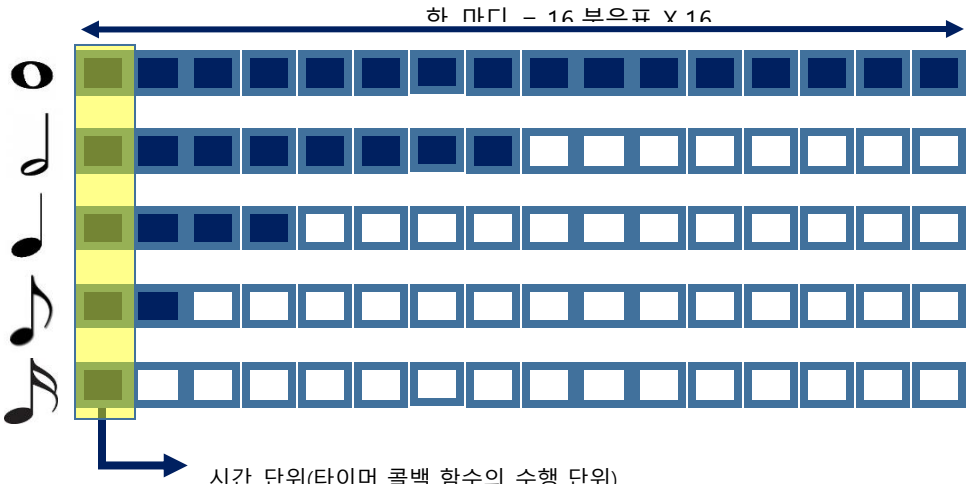
음악의 표현

음악은 음(note)들의 나열이다. 음들은 또한 마디(measure)라는 단위로 묶어진다. 마디는 마디끼리 묶여 선율(melody)을 형성한다. 재밌게도 음악의 기본인 음표는 컴퓨터가 다루기 수월하게 2의 배수로 되어 있다. 아래 그림은 음표와 시간 간격이다.

가장 긴 온음표는 16음표가 16개로 이루어져있다. 이 단위가 한 마디라고 정하겠다. 따라서 32분 음표는 표현하지 않는다. (실제로 잘 안 씀) 점이 붙은 경우는 역시 해당 음표의 절반만큼만 증가하므로 점16분음표를 제외하고는 표현가능하다.

구현 초기 단계에서는 이러한 시간 간격이라는 개념을 노트 on/off 메시지를 통해 구현하려고 했었다. 하지만 마디에 멜로디가 없어 지면서 메모리 공간을 걱정하게 되었다. 시간 간격을 표현하기 위해서는 한 마디의 한 노트가 기본 단위이기 보다는 한 단위에 여러 노트 이벤트들이 존재한다는 것이다. 예를 들어 4분음표가 켜지고 꺼지기 바로 직전에 16분음표가 켜지게 되면 그 다음 시간에는 4분음표를 끄는 명령고 16분음표를 켜는 명령 두 개가 동시에 실행되어야 한다. 이러면 하나의 시간 단위에는 여러개의 노트값을 허용하게 되고 그렇게 되면 마디 하나의 메모리 사이즈는 수배 단위로 증가하게 되어 메모리 관리에 위협적이 된다. 따라서 한 타임 유닛 당 동일 채널에 있는 다른 음들을 전부 오프시키고 새로운 음을 내는 식으로 할 것이다.

미디 라이브러리



<그림 24> 음표 별 시간 할당.

아두이노 미디 명령어 구조가 그렇게 복잡하지는 않지만 만들어준 떡을 먹지 않을 수 없다. 아두이노 미디 라이브러리는 아두이노가 미디 컨트롤러 역할을 할 때 매우 간편한 인터페이스를 제공한다. 아두이노에 기본 내장된 라이브러리는 아니다.

FortySevenEffects / arduino_midi_library

깃에서 위의 이름으로 되어있다. 주로 쓰는 함수는 다음 그림과 같다.

```
53  public:
54      inline void sendNoteOn(DataByte inNoteNumber,
55                             DataByte inVelocity,
56                             Channel inChannel);
57
58      inline void sendNoteOff(DataByte inNoteNumber,
59                              DataByte inVelocity,
60                              Channel inChannel);
61
62      inline void sendProgramChange(DataByte inProgramNumber,
63                                    Channel inChannel);
64
65      inline void sendControlChange(DataByte inControlNumber,
66                                    DataByte inControlValue,
67                                    Channel inChannel);
```

<그림 24> 기본적인 미디 명령 함수 : 위에서부터 각각 노트 온, 오프, 음색 변경, 컨트롤 체인지(볼륨, 리버브, 팬포트와 같은)

타이머 콜백 함수

음악에서 시간을 다루는 것은 필수적이다. 특정 시간이 지나고 다음 음이 울리기 시작하기 때문이다. 앞서 음악을 노트들의 나열이라고 정의했는데 사실 노트들 사이의 침묵의 시간이 있어야 음악이 성립한다. 음악 교과서에 나오는 얘기이다. 프로그래밍에서도 역시 이런 개념을 구현하기 안성맞춤인 개념이 있는데 바로 타이머 콜백 함수이다. 타이머 호출 주기와 하는 일(콜백 함수)를 원하는대로 설정할 수 있다. 여기서는 아두이노에 내장된 하드웨어 타이머를 사용하기 보다 소프트웨어적으로 구현된 타이머를 사용한다. 이 또한 오픈 소스로 작성된 코드이다.

깃에서 위의 이름으로 되어 있다. 여기서 코드 hack이 필요한데 주기 설정은 처음 타이머를 생성할 때만 정하도록 인터페이스가 만들어져 있는데 사실 타이머 밑의 이벤트 함수를 찾고 들어가면 템포를 실시간으로 변경할 수 있다.

```
297  void measure_performer() {
298    static vector<nNote>::iterator curr;
299    // Serial.println("mea per");
300    curr = temp_measure.get_current_tracker();
301    MIDI_BOARD.sendControlChange(AllNotesOff, 127, curr_channel);
302    MIDI_BOARD.sendNoteOn(curr->pitch, 127, curr_channel);
303    temp_measure.inc_note_tracker();
304  }
305
306  void set_measure_tempo(long const tempo) {
307    measure_timer._events[mea_tmr_id].period = tempo;
308  }
```

```
mea_tmr_id = measure_timer.every(curr_tempo, measure_performer);
```

<그림 25> 타이머 콜백 함수와 템포 조절

기능

지금까지 악기의 서비스에 대한 설명은 없었다. 다음은 사용자가 악기를 가지고 할 수 있는 기능들이다.

자유연주

자유연주는 말 그대로 건반을 마음대로 연주할 수 있다. 음색이나 옥타브를 바꾸보면서 음을 조작할 수 있다.

녹음

녹음은 엔코더를 돌리면서 저장할 노트의 위치를 정하고 해당 음을 지정하는 식으로 마디를 만들어 나간다.

재생/중지

재생은 타이머 콜백 함수가 선택된 마디 또는 멜로디를 호출 될 때마다 한 음씩 연주하게 한다. 중지는 재생을 멈추게 한다.

시퀀싱

드럼 머신과 비슷한 개념으로 이것 때문에 led 패널이 있다. 이 기능은 흔히 시퀀서 프로그램에서 보는 것과 비슷하게 기능한다.

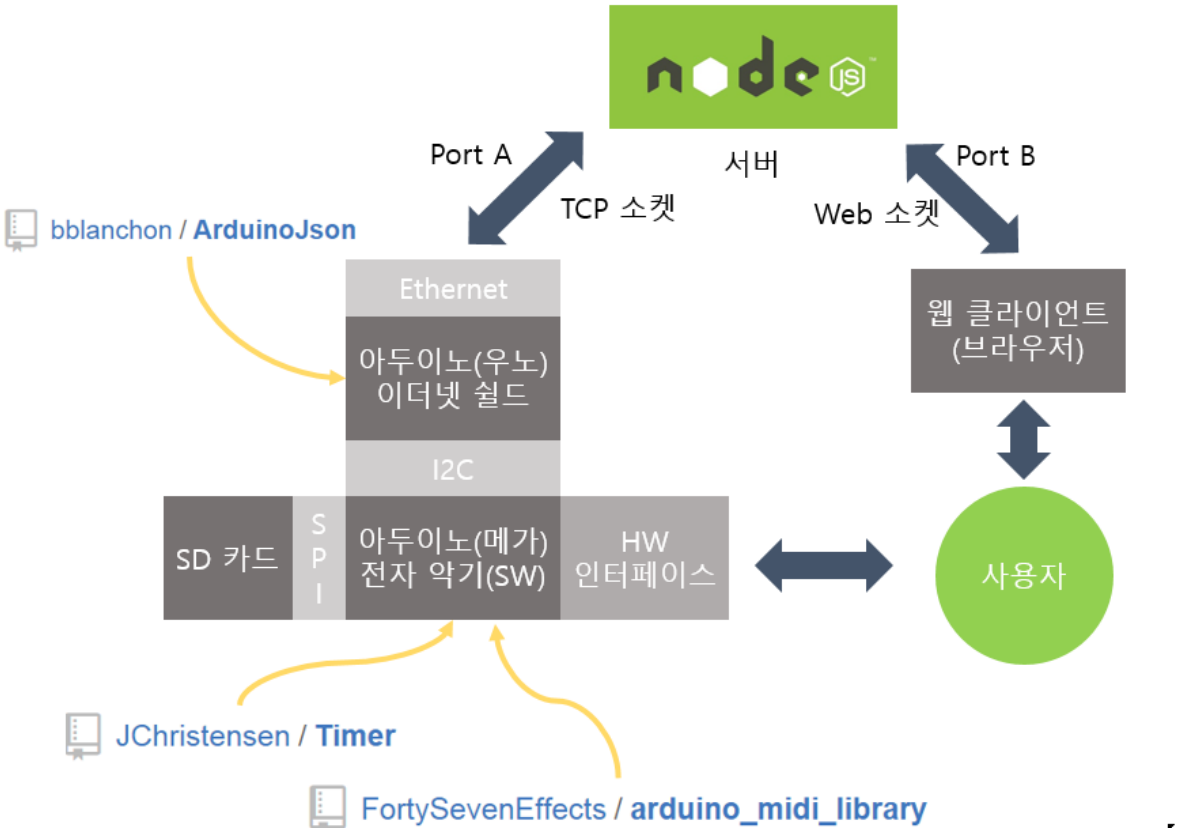
액션들

사용자가 악기에게 입력을 가할 수 있는 리스트는 다음과 같다.

1. 현재 노트 위치 조절
2. 현재 노트 위치 값 삭제
3. 채널 조절
4. 음색 조절
5. 템포 조절
6. 볼륨 조절
7. 리버브 효과 조절
8. 팬포트 효과 조절
9. 마디/멜로디 재생
10. 옥타브 쉬프트(샾)
11. 옥타브 쉬프트(플랫)
12. 다음 마디
13. 이전 마디
14. 마디 추가
15. 마디 삭제
16. 다음 멜로디
17. 이전 멜로디
18. 마디/멜로디 중지

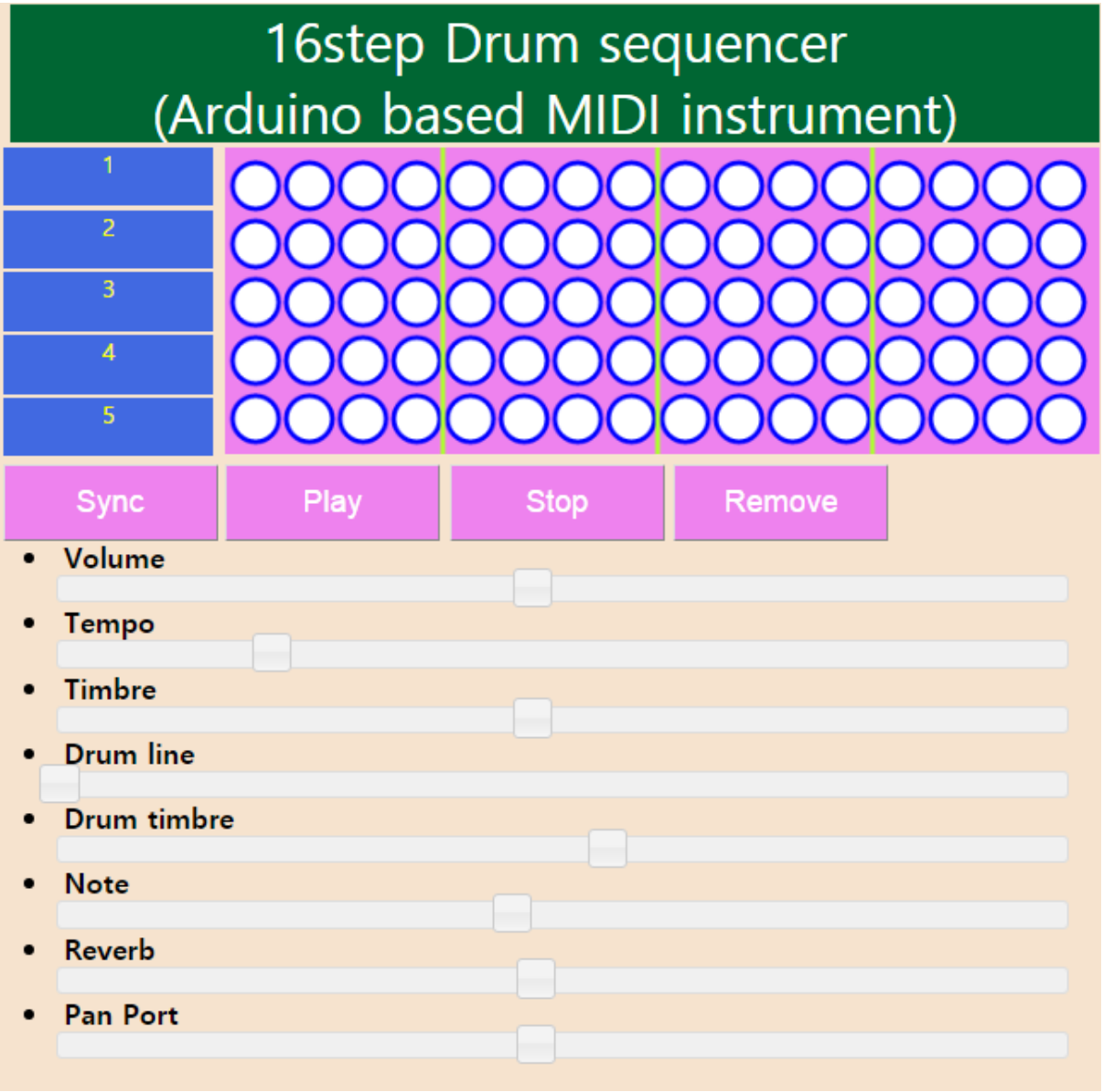
통신

전체 통신 구조는 다음과 같다. Node js로 만들어진 서버가 있고 TCP 소켓으로 통신하는 포트 A와 웹 소켓으로 통신하는 포트 B가 있다. 포트 A는 우노와 통신하고 B는 웹 클라이언트와 통신하게 된다. 전체 그림은 웹 클라이언트 쪽에서 미디와 관련된 몇 가지 값 변화를 보내면 서버가 받아서 우노 쪽으로 보내는 것이다. 이때는 json형식으로 보내기 때문에 받는 우노 쪽에서는 이를 ArduinoJson이라는 오픈 소스를 이용해 파싱한다. 그리고 그 결과를 I2C를 통해 메가 쪽으로 보낸다. 전체 통신 구조 그림은 <그림 26>에 나타나 있다.


Port A
서버
Port B
TCP 소켓
Web 소켓
bblanchon / ArduinoJson
Ethernet
아두이노(우노)
이더넷 쉴드
웹 클라이언트
(브라우저)
I2C
SD 카드
S
P
I
아두이노(메가)
전자 악기(SW)
HW
인터페이스
사용자
JChristensen / Timer
FortySevenEffects / arduino_midi_library

<그림 26> 전체 통신 구조

웹 클라이언트



<그림 27> 웹 클라이언트 아두이노 제어

서버

```
client_server = http.createServer(onRequest).listen(cli_port);
arduino_server = net.createServer().listen(ardu_port);


client_web_socket = io.listen(client_server);
// arduino_tcp_socket = io.listen(arduino_server);

client_web_socket.sockets.on('connection', function(socket){
  //send data to client
  setInterval(function(){
      socket.emit('date', {'date': new Date()});
  }, 1000);

  socket.write("connected to the tcp server\r\n");

  socket.on('client_data', function(data){
    process.stdout.write(data.letter);
  });

  socket.on("music_value_change", function(data) {
    var jstring = JSON.stringify({name : data.name, value : data.valu
    if(ardu_conn == true) {
      sendMsgToUno(jstring);
    } else {
      console.log("it doesn't have receiver.");
    }
    process.stdout.write(data.name + " " + data.value + " " +
                         jstring + "\n\r");
  });
});
```

<그림 28> 서버에서 클라이언트를 위한 웹 소켓을 만듬. 클라이언트에서 넘어온 데이터를 JSON.stringfy로 문자열로 만들어 TCP 소켓인 우노로 전달함.

```
  arduino_server.on('connection', function(socket) {
    asocket = socket;
    ardu_conn = true;
    socket.write("this is new arduino socket communication");
    console.log("connection is made on port " + ardu_port + "#" + arduino_server.con

    // recevie data from arduino and print it
    socket.on('data', function(data) {
      console.log('received on tcp socket : ' + data);
    });

    socket.on('error', function(e) {
      console.log('error event ocurred\n\rarduino might reconnect to this host.');
      socket.write("Reconnecting...");
    });
  });

  //recieve client data

  console.log("Server has started.");
}
function sendMsgToUno(str) {
  asocket.write(str);
}
```

<그림 29> 아두이노를 위한 TCP 소켓 생성. 아두이노가 연결될 때의 소켓을 저장해 두었다가 클라이언트에서 이 소켓에 값을 씀.