QLab 컨트롤러 제작기

글. 김세훈 SBS 미디어IT팀 매니저

서론

SBS 제2뉴스센터(1부조정실)에서는 2010년도에 도입된 AMU 및 DigiCart 등의 오디오 재생 장비를 2024년까지 운용하였다. 그러나 장비의 노후화와 IP 기반 오디오 시스템의 장점 활용을 위해, 2024년 하반기부터 해당 장비들의 교체를 추진하였다. 이 과정에서 DigiCart는 macOS 기반 소프트웨어인 QLab으로 대체되었다.

QLab은 비디오, 오디오, 조명 등을 통합적으로 제어할 수 있는 소프트웨어로, 방송 및 공연 현장에서 광범위하게 사용된다. QLab은 기본적으로 키보드와 마우스를 통한 제어를 제공하나, OSC[1] 프로토콜을 기반으로 한 네트워크 제어도 가능하다. OSC는 실시간 제어를 위한 경량의 통신 프로토콜로, TCP 및 UDP 기반 통신을 모두 지원한다. UDP 사용 시 QLab은 53000 포트로 메시지를 수신하며, 발신자의 53001 포트로 응답을 반환한다. TCP 방식의 경우 OSC 1.1 규격에 따라 double END SLIP[2] 프레임을 사용해야 하며, 송신자 포트로 응답을 반환한다.

본 원고에서는 QLab의 OSC 기반 제어를 목적으로 한 이더넷 지원 MCU 기반 컨트롤러의 개발 과정을 기술한다. 본 컨트롤러는 재생, 정지, 일시 정지, 이전/다음 큐 선택, Fader Start 입력 처리, 주/예비 QLab 선택 제어 기능 등을 지원한다. 구조를 그림으로 표현하면 다음과 같다.

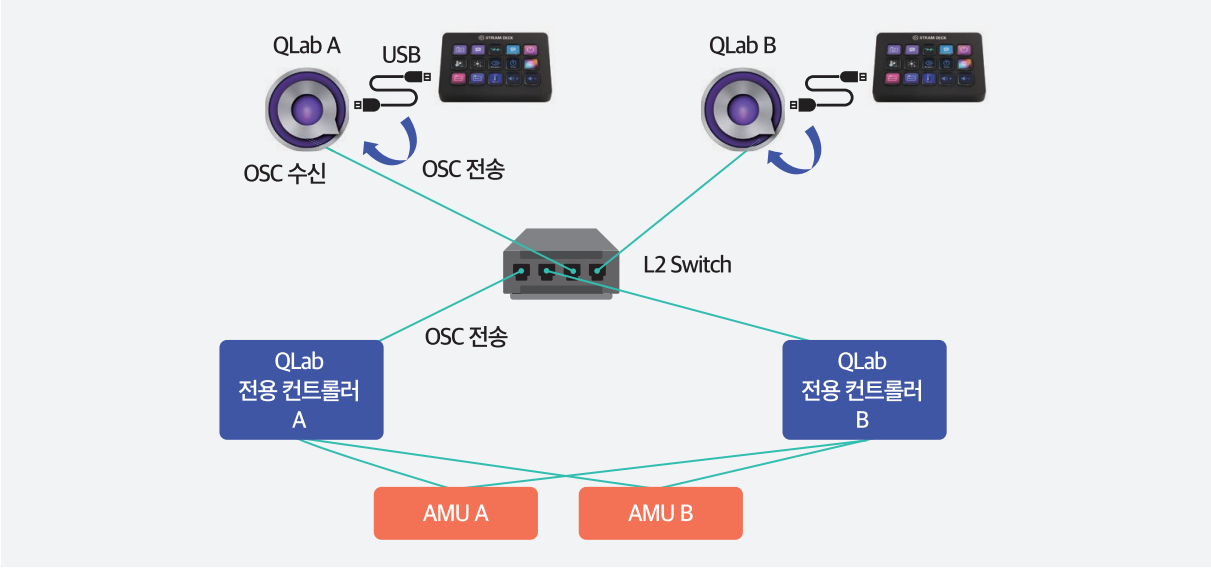

그림 1. 구조도

1. OSC : Open Sound Control, 실시간 오디오 제어를 위한 높은 정확도, 저지연, 경량 특성의 인코딩(포맷)
2. SLIP : Serial Line Internet Protocol, 직렬 통신으로 IP 패킷을 보내는 프로토콜. double END는 데이터 내에 데이터의 끝을 나타내는 문자가 나타날 시 이를 처리하는 방식

전용 컨트롤러 설계

전용 컨트롤러는 L2 스위치를 통해서 네트워크로 QLab이 설치된 mac과 OSC 메시지 통신을 하는 역할을 수행한다. MCU들은 기본적으로 이더넷 기능이 있지 않기 때문에 별도 모듈을 이용하거나, 이더넷 기능이 통합된 보드를 이용하여야 한다.

MCU 선정

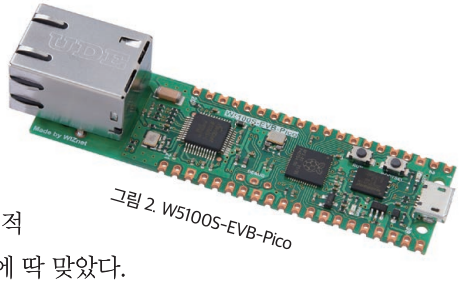

그림 2. W5100S-EVB-Pico

기존 아두이노 보드는 이더넷 기능을 사용하기 위해 별도의 쉴드를 적층해야 하므로 공간 제약이 있는 환경에서 부적합하였다. 이에 따라 위즈네트의 W5100S-EVB-Pico 보드를 채택하였다. 본 보드는 라즈베리파이의 MCU인 RP2040을 기반으로 하며, 이더넷 포트와 W5100S 칩이 통합되어 있다. 물리적 높이가 이더넷 포트 정도로 제한되며, 결과적으로 제공하는 GPIO의 수도 필요에 딱 맞았다.

회로 설계

회로는 키 버튼 보드와 메인보드의 두 부분으로 구성하였다.

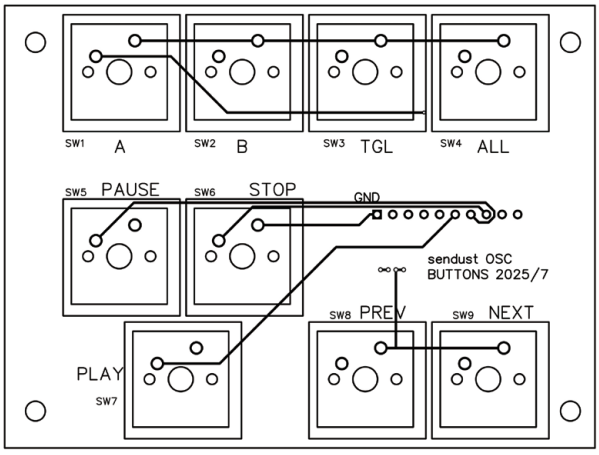

그림 3. 키 버튼 보드

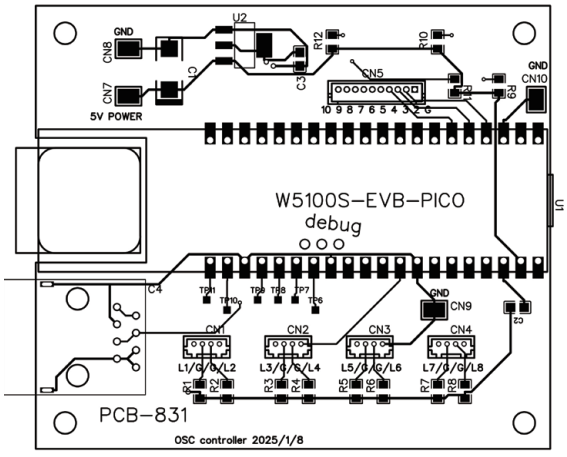

그림 4. 메인보드

키 버튼 보드에는 총 9개의 키 스위치가 배치되었으며, 이는 QLab 제어 관련 A/B/TGL/ALL 버튼 4개와 큐 제어용 버튼 5개(재생, 정지, 일시 정지, 이전, 다음)로 구성된다. 스위치 타입은 체리 갈축을 채택하였고, 키캡은 별도로 주문하였다.
메인보드는 MCU를 중심으로 5V 입력 전원을 3.3V로 낮춰주는 LDO, Fader Start 신호를 입력받는 물리적인 포트인 추가 이더넷 포트, 그리고 이 신호와 MCU를 결합하는 포토커플러 회로 등으로 구성되었다. 메인보드는 MCU와 전원 변환 회로(5V → 3.3V LDO), Fader Start 입력용 이더넷 포트, 포토커플러 회로로 구성된다.
초기 설계에서는 각 버튼에 LED를 부착하여 피드백을 주고자 하였는데, W5100S-EVB-Pico의 GPIO 수 부족으로 인해 I2C 통신을 이용하는 IO 확장 칩 MCP23017의 사용을 고려하였다. 그러나 이후 펌웨어 개발 중에 발생한 문제로 인해 필요 LED 수를 최소화하고 MCP23017 사용은 제외하였다.

PCB 설계

PCB 설계 도구로는 JLCPCB 사의 EasyEDA를 이용하였다. 해당 도구를 이용하면 Schematic 디자인과 선택한 부품의 Footprint 제공을 통한 쉬운 PCB 디자인이 가능하고, PCB 주문 및 조립까지 통합 지원한다. 공휴일을 포함한 짧은 납기와 저렴한 비용으로 빠르게 시제품을 제작할 수 있었다.

펌웨어 개발

개발 기반

RP2040은 microPython/CircuitPython 및 C++ 개발을 지원한다. 초기에는 개발 속도와 메모리 자동 관리의 이점을 고려하여 microPython을 사용하였다. 그러나 상술하였다시피 MCP23017과의 I2C 통신 중 MCU가 멈추는 현상이 반복되는 문제가 있었다. 이에 LED 개수 최소화와 동시에 C++로 포팅하는 작업을 하였고, 결과적으로 MCP23017은 쓰지 않게 되었지만 C++ 포팅도 완료되었기 때문에 C++을 사용하기로 최종 결정하였다.
펌웨어 개발을 위해 hideakitai의 ArduinoOSC, Arduino_JSON, arduino-timer 라이브러리 및 보드 개발사인 위즈네트에서 제공하는 전용 Ethernet 라이브러리를 사용하였다. 그리고 홉 수가 2로 매우 적어 패킷 드랍 가능성이 매우 낮은 환경임과 TCP는 3-Way Handshaking으로 인한 부하와 지연이 더 있다는 것을 고려하여 UDP를 사용하기로 하였다.

기본 기능

큐 이동, 정지, 일시 정지 등의 명령은 QLab이 제공하는 OSC 명령어를 그대로 사용하여 구현하였다. 재생 기능은 ① 현재 선택된 큐의 시점을 질의하고 ② JSON 형태의 응답을 수신하는 이벤트가 발생하면 파싱하여 시점을 저장한 뒤 ③ 기존 재생을 정지한 후 ④ 저장한 시점으로 큐를 로드하여 재생을 시작하는 이벤트 핸들러가 작동하도록 구현하였다. 이때 QLab의 고유한 동작 특성상 정지 후 약간의 시간이 지난 후에 다음 명령들을 보내야 안정적으로 동작함을 발견하였다.
그리고 QLab은 heartbeat 확인 목적으로 thump 메시지를 받으면 그대로 응답해주는 기능이 있어 이를 통해 통신 상태를 주기적으로 확인하여 문제가 있으면 LED가 깜빡이도록 하였다.

Fader Start 기능

Fader Start 신호에 대한 처리는 약간 달랐는데, 뉴스기술팀 오디오 감독님의 요청에 따라 Pre-Fader Level 기능을 이용하여 원하는 시점을 찾아둔 후 Fader를 내리면 그 시점을 저장해 두었다가 실제 송출을 위해 Fader Start 시 그 시점부터 재생하도록 하여 Fader 동작의 편의성을 높였다. 이러한 커스터마이징이 가능하다는 점이 장비를 자체 개발하는 가장 중요한 장점이라고 생각된다.

QLab 선택 기능

이외에도 주(A)/예비(B) QLab 중 어느 쪽을 제어할지를 선택하는 A, B 버튼, QLab을 번갈아 가며 제어하는 기능에 대한 TGL 버튼, 그리고 주/예비 AMU의 Fader Start에 대해 주/예비 상관없이 QLab들을 작동시킬지 결정하는 ALL 버튼 등의 기능을 구현하였다.

Stream Deck 플러그인 개발

Stream Deck은 USB 기반으로 mac과 연결되나 이를 통해 직접 QLab을 제어하는 것이 아니라, macOS에서 실행되는 Node.js 기반 플러그인을 통해 전용 컨트롤러와 같이 OSC 메시지를 네트워크로 송수신한다.

개발 기반

Stream Deck 플러그인은 Node.js 기반으로 제작되며, 언어는 TypeScript를 이용한다. 요구되는 Node.js 버전은 20 이상이다. 그리고 @elgato/cli 패키지를 npm을 이용해서 설치하면 Stream Deck CLI를 이용할 수 있고, 이 CLI 프로그램이 코드를 플러그인으로 컴파일하는 역할을 한다.
의존 패키지로는 node-osc와 osc-js가 있다. 기본적인 OSC 송수신은 node-osc를 이용하고, 큐의 시점 질의 및 재생 시

작 명령을 위한 OSC 메시지 생성을 위해서 osc-js를 이용한다.

플러그인 파일 구조

필자가 제작한 Stream Deck 플러그인은 크게 전역 변수를 저장하는 파일, 플러그인의 액션(버튼 동작)들을 등록/주기적으로 현재 큐 번호를 질의/수신한 응답 처리 핸들러/큐 재생 명령을 송신하는 메인 파일, 그리고 각 버튼의 액션을 정의하는 파일들로 구성되어 있다.

전역 변수를 저장하는 파일에는 주/예비 QLab과의 OSC 통신을 담당하는 각 클라이언트, 클라이언트에 OSC 메시지를 전송하는 함수, 각 QLab의 선택된 현재 큐, 과거 큐를 관리하는 객체, 어느 QLab을 제어할지의 상태를 관리하는 객체들을 담고 있다.

버튼 구성

각 버튼들은 주(A)/예비(B) QLab을 선택하는 A, B, Both 버튼, 전용 하드웨어 컨트롤러와 동일한 재생/정지/일시 정지/이전/다음 큐 선택 버튼, 그리고 큐 번호 버튼으로 구성된다. 큐 버튼은 누르면 해당 큐로 즉시 이동하며, 메인 파일 내의 핸들러에 의해 각 QLab의 선택 상태와 현재 큐 번호에 따라 배경색이 변경된다.

다음은 완성된 QLab 플러그인의 모습이다. 현재 QLab A를 제어하기 위해서 선택한 상태이다. 1페이지에는 기본 기능 버튼들이 있고, 2페이지부터는 큐 번호 버튼들이 있는데, 현재 QLab의 4번 큐가 선택된 상태이기 때문에 해당 큐 버튼의 배경이 빨간색으로 되어 있다.

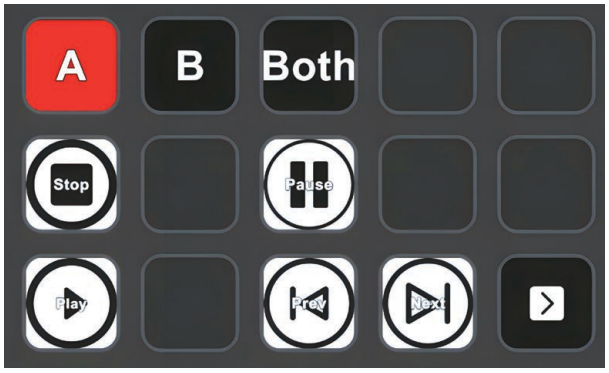

그림 5. Stream Deck 플러그인 1페이지

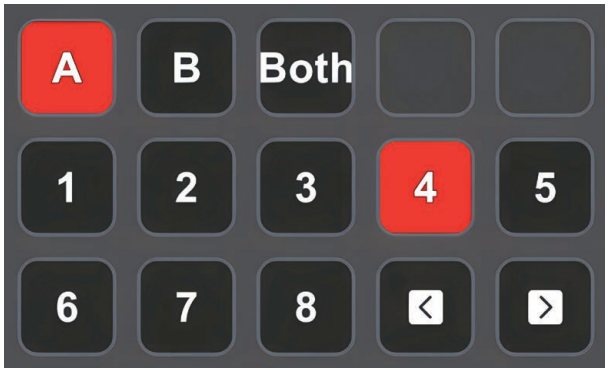

그림 6. Stream Deck 플러그인 2페이지

완성

완성된 QLab 컨트롤러의 외형은 다음과 같다.

그림 7. QLab 컨트롤러 외형

좌측에는 Stream Deck이 있고, 우측에는 키 버튼 보드가 설치되어 있다. Stream Deck의 위쪽에는 QLab A, B와의 통신 상태를 나타내는 LED가 있고, 우측에는

전원 및 A, B, TGL, ALL 상태를 알려주는 LED가 있다.

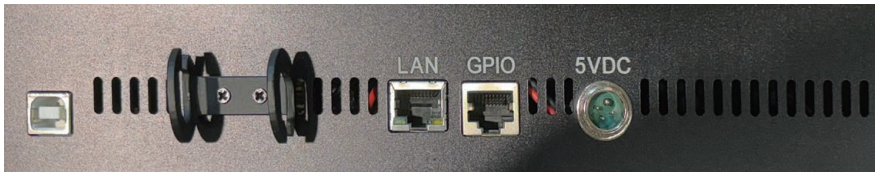

그림 8. QLab 컨트롤러 후면

컨트롤러의 후면은 다음과 같이 전원 및 포트들로 구성되어 있다.
5V 전원은 안전성을 위해 나사로 체결이 가능한 단자를 사용하였고, Fader Start 신호를 위한 포트는 앞서 설명하였듯이 Ethernet 포트를 사용하였다. 좌측의 USB B 포트는 Stream Deck과의 연결을 위하여 간단한 USB A-USB B 컨버터를 만들어서 설치한 것이다.

실제 1부조정실에서는 아래와 같이 QLab 컨트롤러를 설치하여 운용하고 있으며, 주/예비 AMU 각각에 대해 QLab 컨트롤러가 연결되고, 각 QLab 컨트롤러 위에 모니터를 설치하여 QLab 화면을 볼 수 있게 하였다.


그림 9. QLab 실제 설치 모습

어려웠던 점 & 느낀 점

제작 과정에서 몇 차례 어려움을 느낀 지점들이 있다.
첫 번째로는 앞서 언급한 MCP23017과의 연결을 위해 I2C 통신을 microPython으로 구현하였을 때 발생하는 MCU 멈춤 문제였다. 정확한 원인은 알 수 없으나 C++로 하였을 때는 안정적인 것으로 보아, Python의 느린 성능이 영향을 주었을 수 있다고 생각된다.

두 번째로는 Ethernet 라이브러리 사용에 관한 문제였다. 네트워크 연결 유실 시 MCU가 한참 동안 작동하지 않는 현상이 있었다. 왜 그런지 한참 고민해보아도 알 수 없어서 Ethernet 라이브러리 문서를 읽어보니 재전송 관련한 setRetransmissionCount와 setRetransmissionTimeout 메서드가 있음을 알게 되었고, 이 둘을 1로 하였더니 문제가 해소되었다. 문서를 잘 읽어봄으로써 해결할 수 있었으나, 원래 Ethernet은 재전송 기능을 제공하지 않는데, 이러한 함수들이 있는 것은 의아하게 생각된다.

세 번째로는 QLab과의 통신이 간헐적으로 끊기는 현상을 발견하였을 때였다. macOS Sequoia에서는 시스템 설정에서 방화벽 외에도 '로컬 네트워크'라는 기능이 있는데, 여기서 앱별로 통신을 차단/허용할 수 있다. 그런데 QLab의 통신을 허용해두어도, 간헐적으로 QLab의 응답이 송신되지 않는 경우가 발생했다. 이 경우 로컬 네트워크에서 차단/허용 상태

를 다시 설정해주면 해결이 되지만, 언제 이런 상황이 발생할지 예측할 수 없으므로 실사용은 불가능했다. UDP 통신이 문제인 것으로 추정하여 몇 주에 걸쳐 TCP로 변경하였으나 마찬가지였다. 마지막 방법으로 macOS 버전을 낮추려 했으나 도입된 mac이 최신 장비라 OS 다운그레이드가 불가한 난감한 상황이었다. 결국 다른 부서의 구형 mac과 맞교환하여 설치함으로써 불완전하게 문제를 해결하였다. 이 과정에서도 아래와 같이 문서를 읽는 것의 중요성을 다시금 느낄 수 있었다.

먼저, TCP를 이용할 때는 평문을 그대로 보내는 것이 아니라, 앞서 언급한 것처럼 double END SLIP 방식으로 메시지를 프레이밍해야 하는데 처음에는 알지 못했다. 몇 번의 시행착오 끝에 QLab의 문서를 보다 보니 이 부분을 알 수 있었다.

두 번째로 로컬 네트워크 기능이 차단하는 통신의 유형을 공식 문서에서 읽어봤어야 했다. 문서에서는 통신 시작이 인바운드/아웃바운드인지, TCP/UDP인지의 경우에 따라 차단/허용을 설명하고 있었으므로 잘 읽어볼 필요가 있었다.

세 번째 문제는 QLab 5.4.9 버전에서 해결된 것으로 보인다. 릴리즈 노트에 Sequoia에서의 네트워크 문제 관련 변경사항이 있다고 언급되어 있으며, 실제로도 일주일간 지속적인 통신 테스트를 하였는데 문제가 발생하지 않았다. 이에 따라 추후 QLab을 도입해도 안정적 운용에 문제가 없을 것으로 생각하고 있다.

맺음말

이번 프로젝트는 소규모여서 상관이 크게 없으나, 이미 잘 알려진 것과 같이 MCU 프로그래밍은 메모리 최적화 등 성능을 고려하여 프로그래밍할 필요가 있다. 예를 들어 RP2040보다 적은 메모리를 갖는 아두이노 우노를 사용한다면, OSC 수신 메시지를 Event-Driven 방식으로 다루지 말고 동기적으로 폴링할 것을 해당 라이브러리에서 안내하고 있다. 그리고 C++은 다른 고급 언어들과 달리 메모리 쓰레기 수집을 하지 않으므로 동적 변수 생성에 유의할 필요가 있다. 또한 버튼이 눌렸을 때 상태 값을 읽을 시 버튼이 반복적으로 빠르게 붙었다 떨어지는 채터링 현상을 고려하여야 함은 많은 분이 이미 알고 계실 것이다.

필자는 이러한 MCU 프로그래밍을 대학교 때 한번 해본 이후로 처음이라, 이번 프로젝트는 어렴풋하게만 알던 위 사항들을 실제로 고려하고 체화할 수 있었던 소중한 기회였다. 여러 어려움이 있었지만 잘 해결해서 다행이었으며, 또한 문서를 잘 읽어야 함을 다시금 느낄 수 있었다. 그리고 Fader Start 관련 기능처럼 원하는 특수한 기능을 구현할 수 있다는 점에서 자체적으로 장치를 제작하는 것의 장점을 확실히 깨닫게 되었다. 실제로 음악 프로그램을 제작하는 상암, 등촌용으로 코드를 약간 변형한 버전을 추가 개발하여 적용할 수 있었다.

이번 기회를 발판으로 삼아 다음에는 회로 설계, 외형 디자인까지 도전하여 간단한 장치 제작의 전 과정에 필요한 역량을 쌓아서 방송 제작의 편의성과 생산성을 높일 수 있도록 노력을 기울일 것이다.
마지막으로, 이러한 기회를 주시고 문제 해결을 위해 같이 노력해주신 조영훈 선배님, 회로 설계를 해주시고 펌웨어 개발을 믿고 맡겨주셨으며 여러 고민과 조언을 해주신 진신우 부장님, Stream Deck에 관한 제안을 해주신 신규호 선배님, 신현범 차장님, 장비를 실제 사용할 사람의 관점에서 완성도를 높일 수 있도록 도와주신 주호건 선배님께 감사의 말씀을 드리며 글을 마치고자 한다.