

지문인식 결제 시스템

Technical Report

이연준 <yeonfish6040@dimipay.io>



목차

1. 들어가며	4
2. Version 1 초기설계	5
2.1. 요구조건	5
2.2. 지문센서 선정	5
2.2.1. 지문센서 요구사항	5
2.2.2. 지문센서 비교	5
2.3. 하드웨어 설계	6
2.3.1. 하드웨어 요구사항	6
2.3.2. 하드웨어 선정	6
2.3.3. 하드웨어 구성 및 인터페이스 설계	6
2.3.4. 회로 및 PCB 설계	7
2.4. 제품 개발	9
2.4.1. 시스템 구성 및 동작 흐름	9
2.4.2. 알고리즘	10
2.4.2.1. 알고리즘 요구조건	10
2.4.2.2. 알고리즘 선정	10
2.4.3. 백엔드 로직	10
2.4.4. 지문 데이터 캐싱	11
2.4.4.1. 캐시 로딩	11
2.4.4.2. 캐시 갱신	11
2.4.5. BLE 통신 및 키오스크 연동	12
2.4.6. Sensor와 nRF54L15의 통신	12
2.4.7. 하드웨어 프로토타입 주문 및 조립	13
2.4.8. 기능 시험 및 성능 측정	14
2.5. 버전1 제품에 대한 리뷰	14
2.5.1. 인증 지연시간 분석	14
2.5.2. 센서 면적에 따른 인식 실패율 증가	14
2.5.3. PCB 설계 오류	14
2.5.4. 설계 당시 판단 미스	14
2.5.5. 버전 2 에서 고도화가 필요한 사항들	15
3. Version2 완제품	16
3.1. 요구조건	16
3.2. 지문센서의 재선정	16
3.3. 하드웨어 개념 검증(POC)	17
3.4. 하드웨어 설계	18
3.5. 트러블슈팅	20
3.6. 인프라 설계	25
3.6.1. DB 분리	25
3.6.2. 지문 서버 보안	25
3.7. 알고리즘 보완	25

3.7.1. 지문삭제	26
3.8. 펌웨어 설계	27
3.8.1. 지문센서 통신	27
3.8.2. 이미지 처리	27
3.8.3. DFU (Device Firmware Update)	29
3.9. 성능	30
4. 성과	31
5. 참조자료	32

1. 들어가며

본 지문결제시스템은 2026학년도부터 초·중등교육법 제 20조의 5에 근거하여 교내 스마트폰 소지 제한 환경에서도 기존 DIMIPAY 결제 시스템을 사용할 수 있도록 설계되었다. **iPad와의 BLE 연동, 원본 지문 미보관, 2초 이내 인증**을 주요 요구조건으로 설정하였고, 초기 UART 기반 V1은 **평균 인증시간 5.8초와 좁은 센서 면적으로 실사용에 한계**를 보였으며, 병목을 계측한 결과 센서-단말 간 이미지 전송이 전체 지연시간의 약 3분의 2를 차지함을 확인하였다.

이후 HF302GD 센서와 직접 설계한 nRF54L15 보드로 시스템을 재구성하고, **적응형 지문 이진화 알고리즘**을 적용하여 평균 인증시간을 **1.38초로 단축**하였다. 완성된 시스템은 실제 교내 매점에 배포되었으며, 2026년 5월 한 달간 전체 7,454건의 결제 중 3,199건(≈43%)이 지문인식으로 이루어졌다.

시연 영상

V1 프로토타입 ([2.4.8. 기능 시험 및 성능 측정](#))

youtu.be/aYEx_O_Kxps

DFU 무선 펌웨어 업데이트 ([3.8.3. DFU \(Device Firmware Update\)](#))

youtu.be/tb2I9RlrJJ4?t=17

V2 결제 ([3. Version2 완제품](#))

youtu.be/HWvwYAcl2mI?t=73

2. Version 1 초기설계

2.1. 요구조건

- 결제 Kiosk로 iPad를 이용중이었고 하드웨어 개발 후 MFi인증을 받을 수 없어 열려있는 API인 **Bluetooth를 이용해야한다**
- 지문 모듈이 사용자를 인증하는 것이 아닌 이미지를 제공하여 **사용자 인증은 서버에서 진행되어야한다**
- 개인정보 보호법 제 21조에 의거하여 개인정보의 처리 목적 달성 및 개인정보 보유기간의 경과 등 **개인정보가 불필요하게 되었을때는 파기하여야한다**
- 개인정보 보호법 제 29조에 의거하여 개인정보 보관을 이행할때 내부 관리계획을 수립하고 접속기록을 보관하는 등 **기술적 • 관리적 및 물리적 조치를 하여야한다**
- 개인정보보호위원회의 생체정보 보호 가이드라인에 따라, **원본이 아닌 미뉴티아 추출 후 저장, 지문 이미지 전송구간 암호화**가 이루어져야 한다
- 지문을 읽어들이기 시작한 시점으로부터 결제 토큰 발급까지 **가급적 2초 이내에 처리되어야 한다**

2.2. 지문센서 선정

2.2.1. 지문센서 요구사항

원활한 통신 및 인증을 위하여 다음중 하나에 해당하는 센서가 필요했다.

- 시중에서 구매할 수 있는 지문센서중에 USB를 이용하며 **프로토콜을 공개하는 센서**
- UART, I2C, SPI등 **통상적인 MCU가 지원하는 통신 프로토콜을 지원**해야한다.
- 높은 신뢰성을 위하여 지문 유사도 Threshold를 높였을때 원활한 인증을 위하여 **넓은 유효면적이** 있어야한다.

2.2.2. 지문센서 비교

해당 요구조건을 탐색하던 중, 2개의 지문센서를 선정하였다.

제품	인터페이스	유효 캡처면적	역할
HI-LINK ZW0608	UART	12.05*12.05mm ²	키오스크 결제용. MCU와 직접 연결
ZKTeco LIVE20R	USB 2.0	15.24*20.32mm ²	지문 등록용. 프로토콜 비공개로 SDK가 도는 환경(LineageOS)에서만 사용

ZKTeco의 제품으로 모든 지문센서를 대신하고싶었으나, USB 프로토콜을 공개하지 않고 Window, Android, Linux SDK용으로 제공하여 RaspberryPi에 Android를 기반으로하는 LineageOS를 올려 등록용으로 사용하기로 하였고,

Live20R에서 얻은 넓은 이미지로 ZW0608에서 얻은 국소이미지의 Score를 높이려고 시도하였다.

ZKTeco 제품의 usb 프로토콜을 Wireshark로 리버스 엔지니어링하여 내부 구현¹으로 이용하려 하였으나 안정성이 떨어져 파기하였다.

2.3. 하드웨어 설계

2.3.1. 하드웨어 요구사항

- BLE 를 지원할 것.
- 지문 이미지를 서버로 전송할 수 있을 것.
- 원활하게 전원을 공급받을 수 있을 것.
- 단순하게 사용할 수 있을 것

2.3.2. 하드웨어 선정

앞서 요구조건에서 서술하였듯, 지문센서와 아이패드에는 **Bluetooth를 이용하여 통신**하여야 한다. 이를 위하여 Bluetooth가 integrated된 mcu를 사용하는 것이 이상적이라고 판단하여 최근에 배우고있고, BLE가 내장되어있는 **nrf54l15를 사용하기로 하였다.**

하지만 nrf54l15보드를 설계하기에는 당장 3학년 1학기 개학이 2주 채 남지 않은 시점에 교내 인트라넷 개발 마감이 최우선순위이어서 시간이 부족한 관계로 **MINEWSEMI사의 nRF54L15-ME54BS01 모듈**을 이용하여 설계하였다.

2.3.3. 하드웨어 구성 및 인터페이스 설계

보드가 외부와 접하는 인터페이스는 다음 3개이다. 전원을 받는 USB Type-C 커넥터, 지문센서가 물리는 6핀 FPC 커넥터(J1), 그리고 펌웨어 기록과 디버깅에 쓰는 10핀 SWD 소켓(J2)이다.

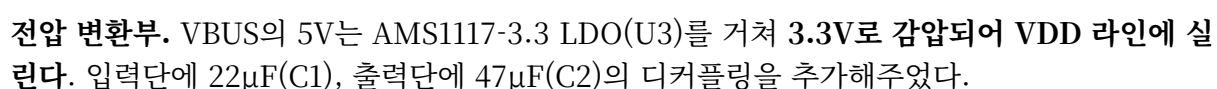
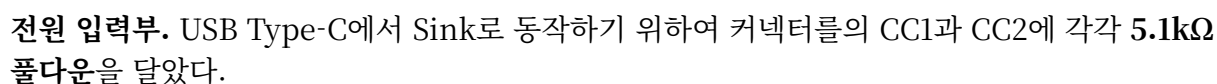
J1 핀	네트명	MCU 핀	역할
1	FP_GND	—	센서 접지
2	FP_RXD	P1.06 (9번)	MCU TX → 센서 RX
3	FP_TXD	P1.04 (12번)	센서 TX → MCU RX
4	FP_VDD_3.3V	—	센서 로직 전원
5	FP_Detect	P1.14 (7번)	손가락 감지 인터럽트
6	FP_SENSOR_3.3V	—	센서 구동 전원

UART는 한쪽의 TX가 상대의 RX로 들어가야 하므로, 회로도에서는 FP_TX를 FP_RXD에, FP_RX를 FP_TXD에 잇는 넷 별칭을 따로 두어 이 교차를 명시하였다. 다만 이 교차는 실제 주문한 보드에서 의도대로 반영되지 않았고, 그 경위는 [2.5.3. PCB 설계 오류](#) 에서 다룬다.

FP_Detect를 P1.14에 배치하여 인터럽트를 받아 MCU가 깨어나 iPad에 알리고, 실제 캡처 여부는 키오스크가 결정하도록 역할을 나누었다. 이 라인은 Active Low이며, 펌웨어는 양쪽 엣지를 모두 구독하여 접촉과 이탈을 각각 별도의 이벤트로 올려보낸다([2.4.1. 시스템 구성 및 동작 흐름](#)).

SWD 소켓은 10핀 IDC 커넥터에 SWDCLK, SWDIO, nRESET을 그대로 뽑고 VTref를 MCU_VCC에 물렸다.

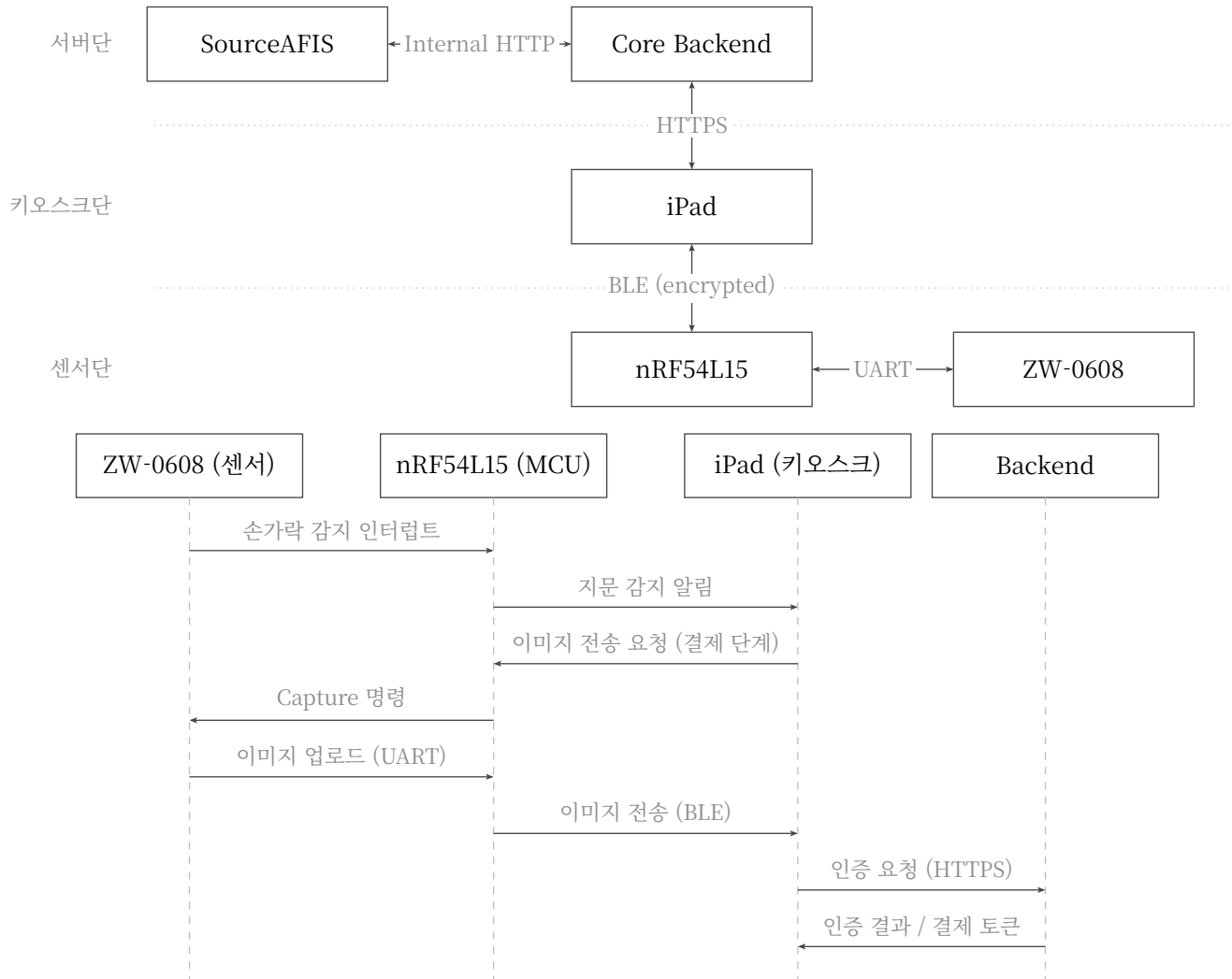
회로는 전원 입력부, 전압 변환부, MCU, 그리고 두 개의 커넥터 블록으로 이루어진다.



2.4. 제품 개발

2.4.1. 시스템 구성 및 동작 흐름

통신 구조는 다음과 같다.



앞에서 서술했듯 ZW-0608의 손가락 인터럽트 라인을 nrf54l15에서 구독하여 인터럽트가 발생하면 ipad의 키오스크 앱에 해당 사실을 알린다.

키오스크는 해당 이벤트를 받았을때 상품 결제단계라면 컨트롤러의 지문결제 함수를 호출하게 된다.

이후 이미지 전송 요청을 발송하여 nrf54l15가 zw-0608에 이미지 capture 이후 버퍼에 있는 이미지를 nrf54l15에 업로드한다 해당 이미지를 다시 nrf54l15가 ipad로 전송하여 ipad는 이 이미지를 서버에 전송하여 서버가 신원을 확인하게된다.

2.4.2. 알고리즘

2.4.2.1. 알고리즘 요구조건

신뢰성있는 결제 시스템 구성을 위하여 요구되는 사항은 다음과 같다

- 신뢰성이 입증된 알고리즘일 것
- 템플릿 추출이 가능할 것
- 빠른 매칭이 될 것

2.4.2.2. 알고리즘 선정

그래서 다음 2개를 한번 두고 비교해보았다.

알고리즘	구현	검토 내용	결과
NFIS (NIST)	C	FBI가 사용한다고 알려져있다. C로 작성되어 매우 빠르며 템플릿 저장도 가능하다. 다만 코드를 직접 빌드해야 하는데 레거시이기에 빌드 오류가 많았고 직접 고치기에는 코드량이 너무 많아 도커로 레거시 시스템에서 빌드하여 테스트하였다. 지원하는 최소 이미지 크기가 ZW0608이 제공하는 이미지보다 커서 사용할 수 없었다.	반려
SourceAFIS	Java / .NET	Score 조정으로 FAR를 근사치로 직접 조정할 수 있고 템플릿을 따로 저장 가능하다. 매칭 시간도 굉장히 짧아 구상 중인 프로젝트에 이상적이었다. 다만 공식 구현체가 Java와 .NET 뿐이라 Bun 런타임 기반의 백엔드에서 직접 사용할 수 없었다.	채택

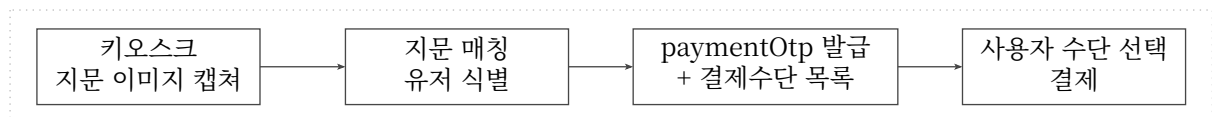
JS로 재구현하자기에는 깡 연산이 JAVA보다 느리고 포팅에 드는 시간과 노력이 너무 많기에 **JAVA 웹서버를 하나 더 띄워서 내부망에서만 접근 가능하게 구성하는 것으로 대체하였다.**

이러한 이유로 결국 SourceAFIS로 알고리즘을 선정하여 지문 비교, 등록, 삭제를 지원하는 웹서버를 만들어서 내부망에 배포하였다. Core Backend는 이 서버를 Internal HTTP로 호출하며, **지문 이미지 데이터가 외부망을 뺄 것으로 경유하는 경로는 존재하지 않는다.**

2.4.3. 백엔드 로직

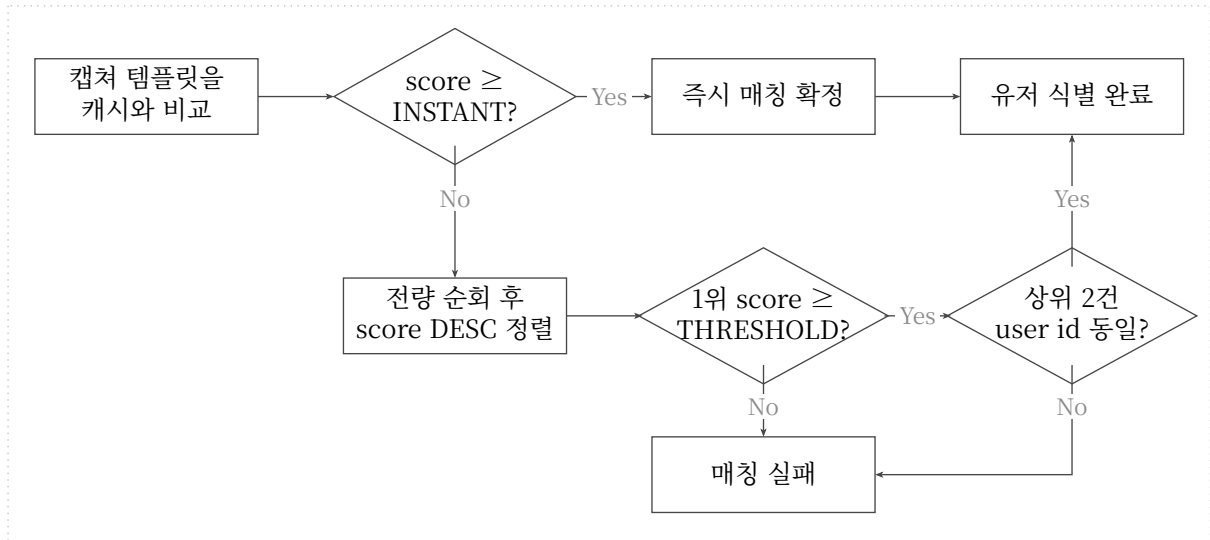
지문 인증을 통한 결제 플로우는 다음과 같다

결제 플로우



키오스크가 지문이지지를 캡처하여 서버에 발송하면 **매칭을 진행하여 유저를 식별**하고, 유저가 식별되면 **paymentOtp를 발급**하여 사용 가능한 결제 수단과 함께 키오스크로 반환한다. 최종적으로 키오스크에서 사용자가 결제수단을 선택하면 **그 결제 수단으로 결제를** 수행한다.

매칭 판정



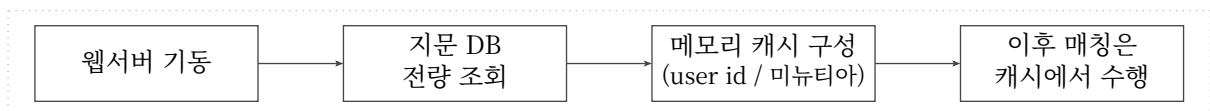
매칭시 빠른 매칭을 위하여 MATCH_INSTANT_THRESHOLD를 설정하고 해당 threshold를 넘으면 즉각적으로 매칭되고 일반적으로 모든 지문을 순회하고 DESC 정렬한 다음 MATCH_THRESHOLD를 넘으며 상위 2개의 지문의 user id가 같다면 매칭에 성공한다. 아무래도 결제 시스템이기에 FAR 민감하여 혹시 모르니까 추가한 로직이다.

2.4.4. 지문 데이터 캐싱

지문 템플릿은 DB에 저장하여 관리하여야하고, 당연히 매 조회마다 DB에 쿼리를 날려서 비교할 수는 없으니, 캐싱을 진행하였다.

2.4.4.1. 캐시 로딩

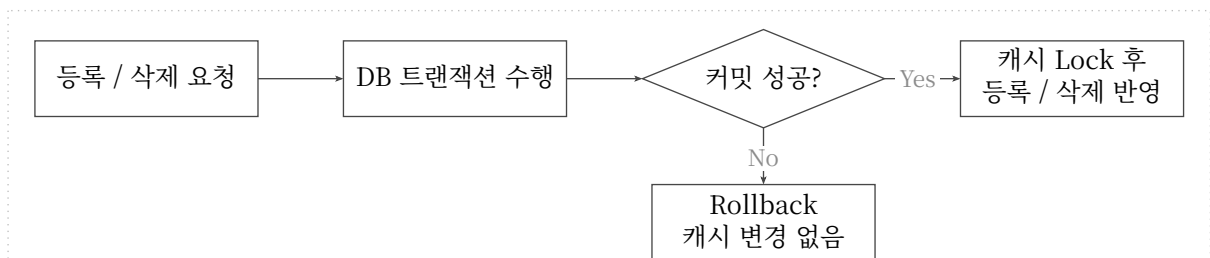
웹서버 기동 시



초기 웹서버 실행시 지문 데이터를 전부 조회하여 user id 따로, 미뉴티아 데이터 따로 메모리에 저장하여 해당 캐시에서 비교를 수행하였다.

2.4.4.2. 캐시 갱신

등록 / 삭제



런타임에서의 지문 DB 업데이트에는 DB 트랜잭션 수행 후 수행 결과에 따라 cache에 락을 건 이후 등록 혹은 삭제 작업을 수행하여 DB와의 안전한 데이터 동기화를 진행하였다.

캐시가 DB보다 앞서거는 상황을 만들지 않기 위하여, 캐시 반영은 항상 DB 트랜잭션이 커밋된 이후에만 수행된다.

2.4.5. BLE 통신 및 키오스크 연동

nRF54L15와 iPad는 BLE로 연결된다. MCU가 올려보내는 것은 손가락 감지 알림과 지문 이미지 두 가지이고, 키오스크가 내려보내는 것은 세션 키 설정과 이미지 전송 요청이다.

- **전송 계층.** 별도 GATT 프로파일을 정의하는 대신 Nordic UART Service를 그대로 쓰고, 초기에는 그 위에 hex 문자열로 전송하였다. 개발 중 로그를 그대로 읽을 수 있다는 점을 우선한 선택이었다. V1이 프로덕션 레벨까지 올라오지 못했으니 해당 프로토콜이 나중에 수정되지는 않았다. 개발 이후에 생각해보면 어짜피 hex도 변환해서 봐야하는데 편하게 할거면 ASCII 문자를 OPCODE로 이용하는 편이 더 디버깅할때는 효율적일 수 있었을 것 같다.
- **세션 키 교환.** 지문 이미지는 BLE 본딩이나 링크 암호화에 기대지 않고 **응용 계층에서 직접 암호화** 한다. 링크 암호화만 믿으면 보호 범위가 MCU-iPad 무선 구간에만 한정되고 그마저도 페어링 상태에 의존하는데, 지문은 유출되어도 바꿀 수 없는 데이터이므로 전송 경로의 성질과 무관하게 데이터 자체가 보호되어야 한다고 판단하였다.

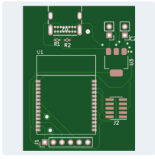
절차는 다음과 같다. 연결되면 키오스크가 서버의 RSA 공개키를 올려보내고, MCU는 자신의 AES 키를 해당 공개키로 암호화하여 서버에 보고한다. 이후 해당 키로 지문이미지를 올려보낸다. AES로 전환하는 이유는 RSA, AES 둘다 Arm CryptoCell에 의해 가속되나 AES의 속도가 더 빨랐기 때문이다.

2.4.6. Sensor와 nRF54L15의 통신

- **baudrate 순회.** 센서 출하값은 9600이었는데, 쓰다가 57600으로 올리고 다시 115200이 가능하다는 것을 알고 또 올리다 보니 손에 있는 센서마다 속도가 제각각이 되었다. 그래서 부팅 시 후보 속도를 차례로 시도해 응답이 돌아오는 속도를 실제 속도로 확정하도록 하였다. 여기서 **핸드셰이크가 0이 아닌 확인코드를 돌려주는데도 UART 링크 자체는 밀접한 현상**을 발견하였다. 확인코드만 보고 링크 불량으로 판정하면 정상 속도를 지나쳐 결국 전 속도에서 실패하게 되므로, 핸드셰이크가 실패하면 같은 속도에서 ReadSysPara를 한 번 더 보내 그 응답이 정상이면 채택하도록 fallback을 추가했다.
- **원본 프레임 정렬.** 센서가 올려보내는 스트림은 유효 화소만 담긴 형태가 아니라 **144행 × 256바이트로 패딩된 36,864바이트 프레임** 이었다. 지문이 담긴 영역은 32~111행의 80개 행이고, 각 행 안에서도 두 블록으로 나뉘어 있으며 이 두 블록이 각각 출력 이미지의 짝수 행과 홀수 행에 해당한다. 이 배치는 데이터시트에 없어 원본 덤프를 직접 열어 화소 분포를 보고 알아냈다. 펌웨어는 이 규칙대로 프레임을 잘라 붙여 **160 × 160, 4bpp** 의 12,800바이트 이미지로 만든 뒤 BLE로 넘긴다. 센서가 앞뒤에 왜 빈 더미 값을 보내는지는 아직까지도 잘 모르겠다.
- **소요 시간의 하한.** UART 구간이 오래 걸리는 원인은 계산으로도 그대로 나온다. 8N1에서 한 바이트는 시작·정지 비트를 포함해 10비트이므로, 36,864바이트를 115,200 bps로 받으면 $36864 \times 10 / 115200 \approx 3.2$ 초다. 여기에 패킷 오버헤드와 재시도가 더해져 **2.4.8. 기능 시험 및 성능 측정** 에서 측정된 **평균 3.8초** 가 된다. 즉 이 구간은 구현이 비효율적이어서 느린 것이 아니라 **전송해야 할 바이트 수와 회선 속도가 정해 놓은 한계**에 가깝고, 그래서 **2.5.1. 인증 지연시간 분석** 에서 보듯 코드를 다듬는 것으로는 줄일 수 없다.

2.4.7. 하드웨어 프로토타입 주문 및 조립

2026-02-17 22:33:57 | W2026021721336894 | PCB

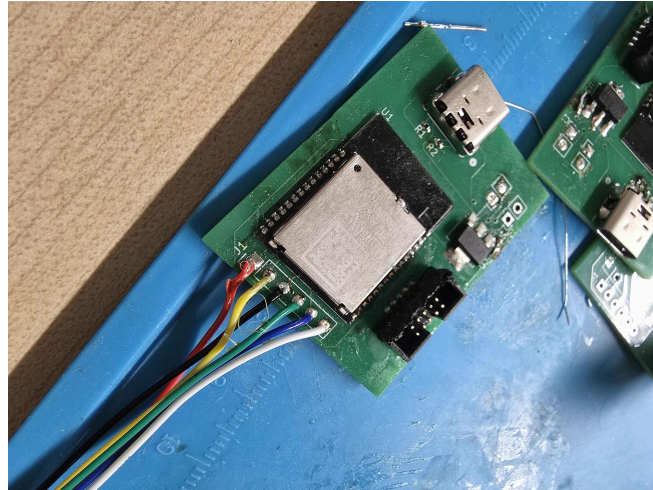


Merchandise Total: \$10.50
Shipping Charge: \$15.50
Order Total: \$26.00

[Shipped](#)
DHL Express
[Shipment Tracking](#)

[Reorder](#)
[Order Details](#)
[Invoice](#)
[Write a Review](#)

[Show All](#)



앞서 말했듯, 당초에는 보드를 직접 식각하여 수제작할 계획이었으나, 부품 실장 난이도와 남은 일정을 고려하여 주문 제작으로 방향을 바꾸었다. 앞서 언급한 via 지름이 수제작 기준으로 남아 있게 된 것도 이 전환 과정에서 설계를 다시 검토하지 않았기 때문이다.

이후 Aliexpress에서 부품을 구매하여 실장하였다.

수령한 보드에서는 UART 배선과 LDO 접지 두 곳을 손으로 수정하여 동작시켰다. 두 문제의 원인과 조치 내용은 [2.5.3. PCB 설계 오류](#) 에서 서술한다.



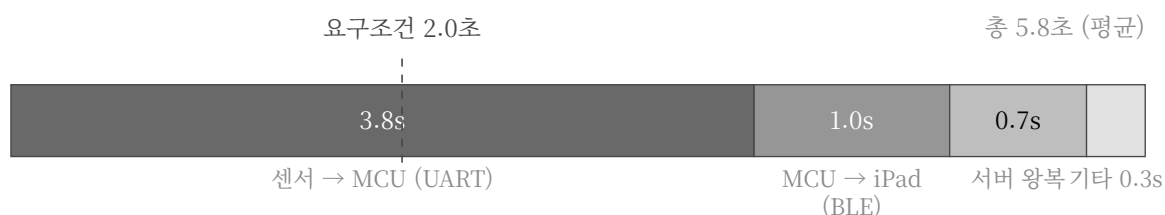
위는 3D 프린터로 제작한 케이싱을 입힌 프로토타입 기기이다.

2.4.8. 기능 시험 및 성능 측정

보드·센서·iPad·백엔드를 모두 연결하여 지문만으로 결제가 끝까지 성사되는 것을 확인하였다. 실제 동작은 [시연 영상²](#)에서 확인할 수 있다.

이후 요구조건으로 걸어진 **2초 이내 인증**을 만족하는지 확인하기 위하여 지문을 올린 시점부터 결제 토큰이 발급되기까지의 시간을 구간별로 나누어 측정하였다. ~~시연할때부터 측정에~~ ~~기차도 없을만큼 느리긴 했다~~

시연 영상³에서 동작을 확인해볼 수 있다.



서버 왕복은 Cloudflare 프록시를 포함한 600~800ms, 기타는 캡처 및 전처리 구간이다.

평균적으로 인증이 완료되는데 **5.8초가 걸렸다**. 시간을 세세히 분해해본 결과 **이미지를 센서에 업로드하는데 평균 3.8초**, BLE로 전송되는데 **평균 1초**, 서버에 전송해서 클라우드 플레이어 프록시를 통하여 응답이 돌아오는데 **600~800ms** 정도로 오랜시간이 걸렸다.

2.5. 버전1 제품에 대한 리뷰

2.5.1. 인증 지연시간 분석

[2.4.8. 기능 시험 및 성능 측정](#)의 측정 결과를 구간별로 놓고 보면 병목이 어디인지는 분명하다. 즉 **전체 시간의 약 3분의 2를 UART 업로드 한 구간이 잡아먹고 있었다**. 전송단계에서 압축 또는 최적화를 통하여 최대한 전송속도를 확보한다고 해도, **uart baud rate제한으로 3.8초 이하로는 전송속도를 개선할 수 없었다**.

2.5.2. 센서 면적에 따른 인식 실패율 증가

Live20R로 얻은 큰 이미지 덕분에 평균적인 매칭 스코어는 50점내외로 올릴 수 있었으나 사람마다 지문 퀄리티의 차이가 있어서 **균일한 정확도를 확보하기 어려웠고**, 상기한 느린 속도로 인하여 여러번의 재시도를 통한 결제가 **빠른 순환이 필요한 점심/저녁의 피크시간대의 매점에서는 현실적으로 어려웠다**.

2.5.3. PCB 설계 오류

UART의 RX와 TX를 반대로 매핑하여 주문하였다. 간단하게 센서와 보드가 연결되는 선을 바꿔 끼움으로써 해결할 수 있었다.

2 layer pcb에서 back layer에 gnd pour을 하지 않은 상태에서 VBUS 5v에서 VDD 3.3v로 바뀌는 LDO 전원블록에서 gnd를 이어주지 않았었다. Gnd Pour를 하지 않고 gnd via만 추가하여서 생겼던 문제였다.... 손납땜으로 gnd를 연결해주어서 해결하였다.

2.5.4. 설계 당시 판단 미스

여러 생각을 했다. 우선 원래는 dk에서 zw-0901로 테스트를 완료하고 급하게 PCB를 시키고 테스트를 하던 도중에 다른 사람 지문에서는 인식이 떨어져서 ZW-0901에서 ZW-0608로 급하게 변경한터라 전송속도를 테스트해볼 수 없었고, 데이터양으로 전송속도를 가늠할 수 있었지만

트러블슈팅 과정이었기에 여러가지를 따져보지 못하고 급하게 때우게되어 문제가 발생했던게 참 안타까웠다.

또한 PCB 설계시 IFA 근처에 KEEP OUT영역이 필요한데 해당 KEEP out이 지켜지지 않았고, 안테나 바로 아래에 3.3V 전원라인이 지나갔다. 참으로 제대로 된게 없는 설계였으나 불행중 다행으로? GND pour를 잊어버려서 RF성능이 크게 떨어지는 참사는 일어나지 않았다.

심지어 PCB를 수제작할 생각이었지만 주문생산하게 되어, 수제작에 맞춰 디자인한 VIA 크기가 PCB 주문제작에 그대로 유지되어 비아크기가 비정상적으로 커졌다. 보기 좋은 설계는 아니다.

조금만 더 여유있게 제작했다면 프로토타입을 좀 더 완성도있게 구성할 수 있었을텐데라는 아쉬움이 있었다

2.5.5. 버전 2 에서 고도화가 필요한 사항들

앞에서 확인한 문제들을 Version 2의 과제로 옮기면 다음과 같다.

1. **센서-MCU 전송 인터페이스 교체.** 지연시간의 3분의 2를 차지한 UART 업로드는 baud rate 상한 때문에 최적화로 줄일 수 없다. 따라서 요구조건 자체에 **12MHz 이상의 SPI 통신을 지원하는 지문센서**라는 조건을 추가하여, 센서 선정 단계에서부터 이 병목을 배제한다.
2. **센서 유효면적과 해상도 확대.** 국소 이미지로 인한 인식 실패율 증가는 threshold를 낮추는 것으로 대응할 수 없으므로(그만큼 FAR가 올라간다) 센서 자체를 바꾸어야 한다. 시중 유통 제품으로는 한계가 있어 제조업체를 직접 컨택하는 방향을 검토한다.
3. **전송량 자체의 감소.** 인터페이스를 바꾸더라도 이미지를 원본 그대로 올리면 BLE 구간이 다 음 병목이 된다. 센서가 주는 8bpp 이미지를 MCU에서 전처리하여 전송량을 줄이는 방안이 필요하다.
4. **현장 운영성 확보.** Version 1은 페어링 상태나 오류를 확인할 방법이 SWD 접속뿐이었다. 매점 상주 인원이 개발자 없이도 상태를 확인하고 조치할 수 있는 수단과, 분해 없이 펌웨어를 갱신할 수 있는 경로가 필요하다.

이 4개의 문제점이 Version 2의 출발점이 되었다.

3. Version2 완제품

3.1. 요구조건

요구조건은 V1에서 “12MHz 이상의 SPI통신을 지원하는 지문센서”라는 조건이 추가되었다.

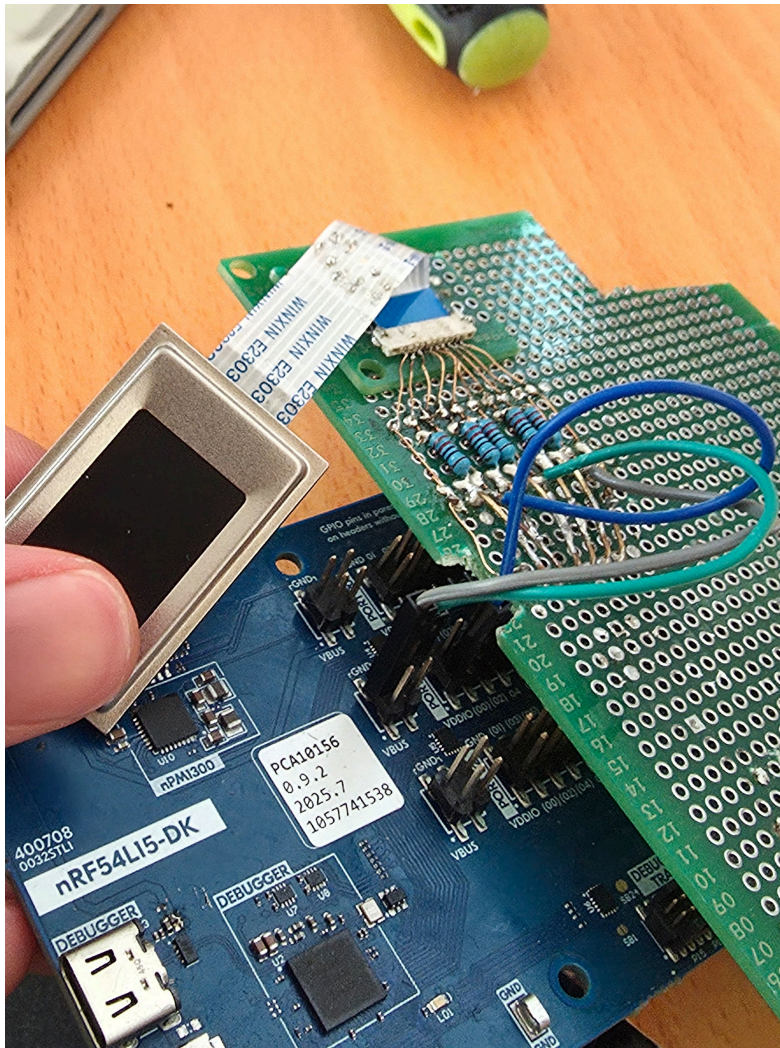
3.2. 지문센서의 재선정

시중에서 구매 가능한 지문센서로는 한계가 있다는 것을 확인하고 지문센서 제조업체를 찾기 시작하였다

제조사 / 모델	인터페이스	이미지	검토 결과
CAMABIO	UART, USB	—	센서 자체는 이상적이었으나 SPI를 지원하지 않고 USB 프로토콜도 공개되어 있지 않아 탈락
Midas Touch MFC-1208GD	—	HF302GD보다 적음	픽셀수가 부족하여 탈락
Aratek A400-M	—	—	동작 전압이 MCU와 일치하지 않아 추가적인 레벨 변환 설계가 필요하여 탈락
ADH-TECH HF302GD	SPI 16MHz	256*360, 508dpi 최대 9fps	넓은 센서면적과 높은 해상도, 빠른 캡처 속도로 채택

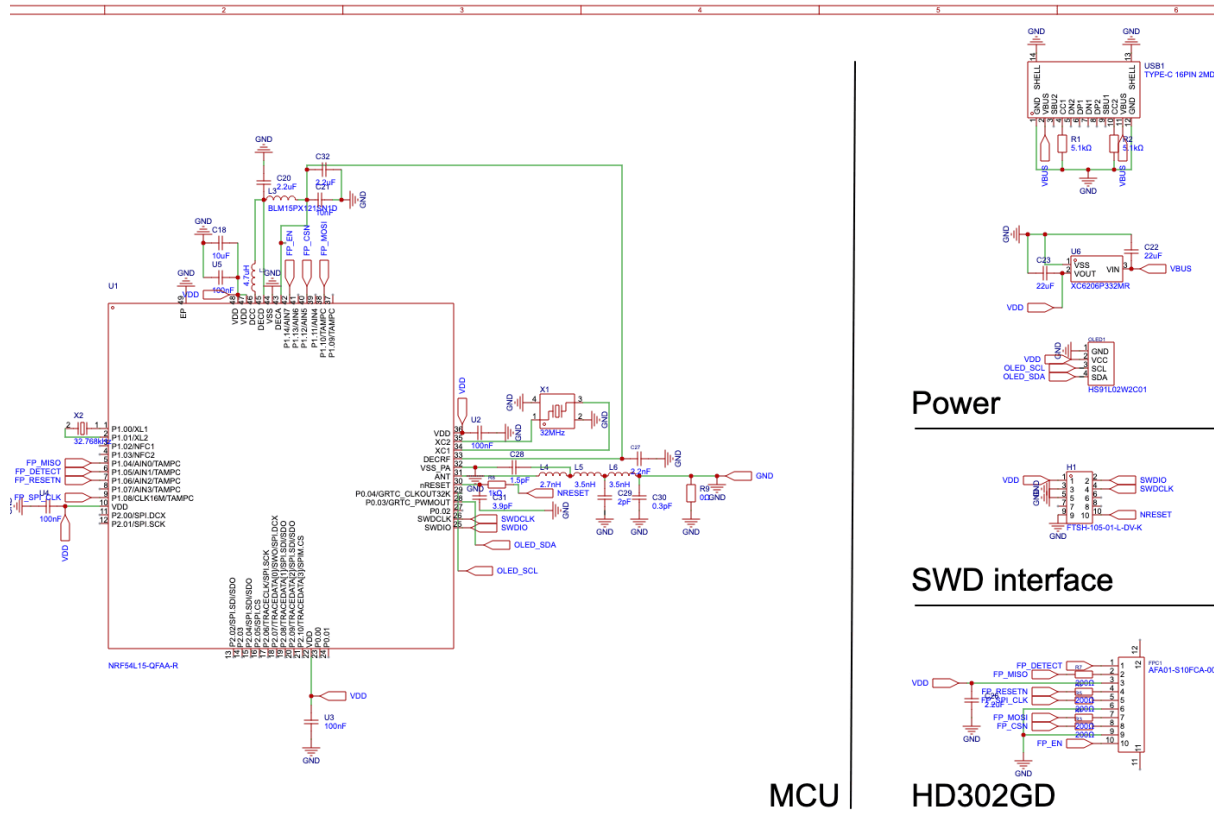
ADH-TECH사의 HF302GD가 최종적으로 선택되었다. **SPI 16MHz 통신**이 가능하고 지문 이미지를 받을 수 있으며, **256*360에 508 dpi**로 넓은 센서면적과 높은 해상도로 적합하였다. 지문 캡처도 최대 9fps로 빠른 촬영이 가능했다.

3.3. 하드웨어 개념 검증(POC)



nRF54L15-dk를 이용하여 하드웨어 POC를 제작하였다. ADH-TECH사에 이메일로 컨택하여 하드웨어 샘플과 테스트보드를 구매하여 수령한 이후, 테스트보드에서 FPC 커넥터를 뽑아다가 만능기판에 납땜하여 전원과 SPI 데이터라인을 구성하여 nRF54L15-dk에 연결하여 하드웨어 POC를 제작하였다.

3.4. 하드웨어 설계

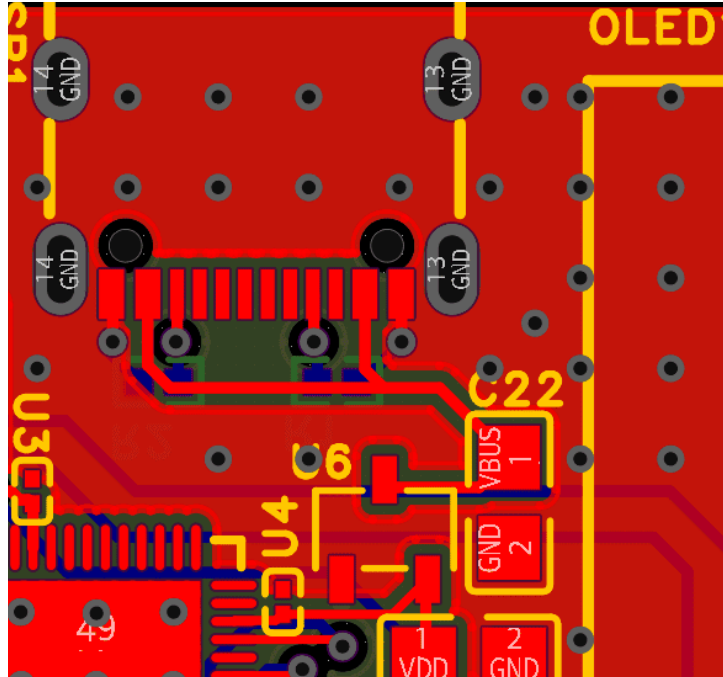


해당 시점에는 시간과 예산이 충분한 관계로 nRF54L15 보드를 직접 설계하여 pcba를 맡기기로 하였다. 또한 EasyEDA를 이용하면 주문시 할인이 되기에 원래 사용하던 Kicad가 아니라 EasyEDA로 설계하게 되었다. Nordic semiconductor사의 nRF54L15 데이터시트에 나와있는 Reference Circuitry⁴를 참고하여 디자인하였다. 전원부와 필터, RF 서킷은 신뢰성있는 동작을 보장하기 위하여 해당 데이터시트에 있는 BOM 스펙을 그대로 차용하여 사용하였다.

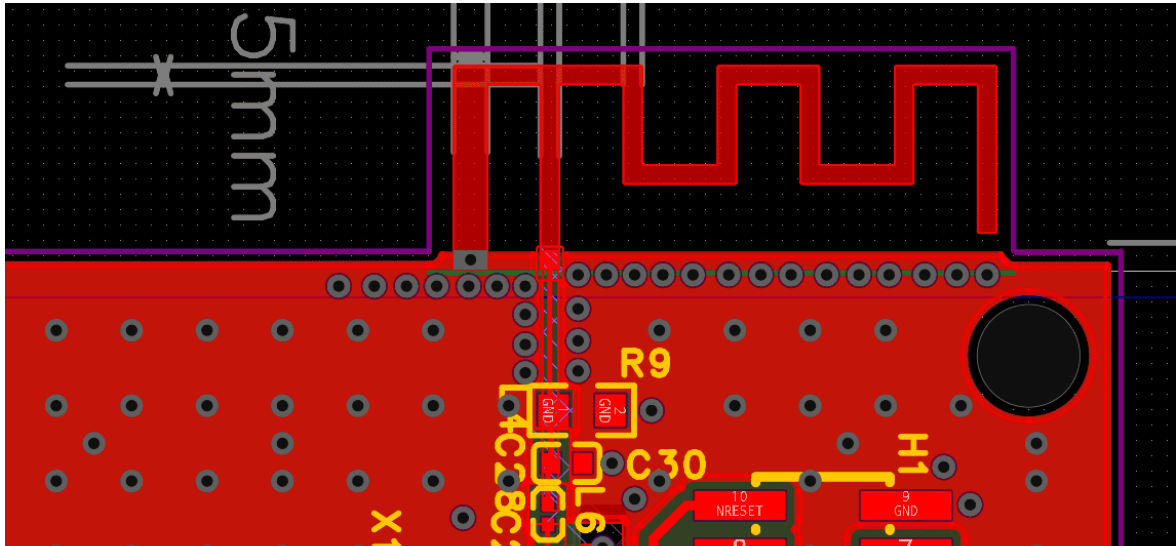
지문센서와 MCU간의 SPI 통신라인에는 지문센서 제조사측 데이터시트에 기술된 링잉 억제용 2000ohm의 저항을 넣었다.

Kicad와 다르게 JLCPCB Components Lib에서 바로 부품을 검색해서 넣는 것이 가능했기에 BOM을 추가로 작성해주지 않아도 되어 간편했다

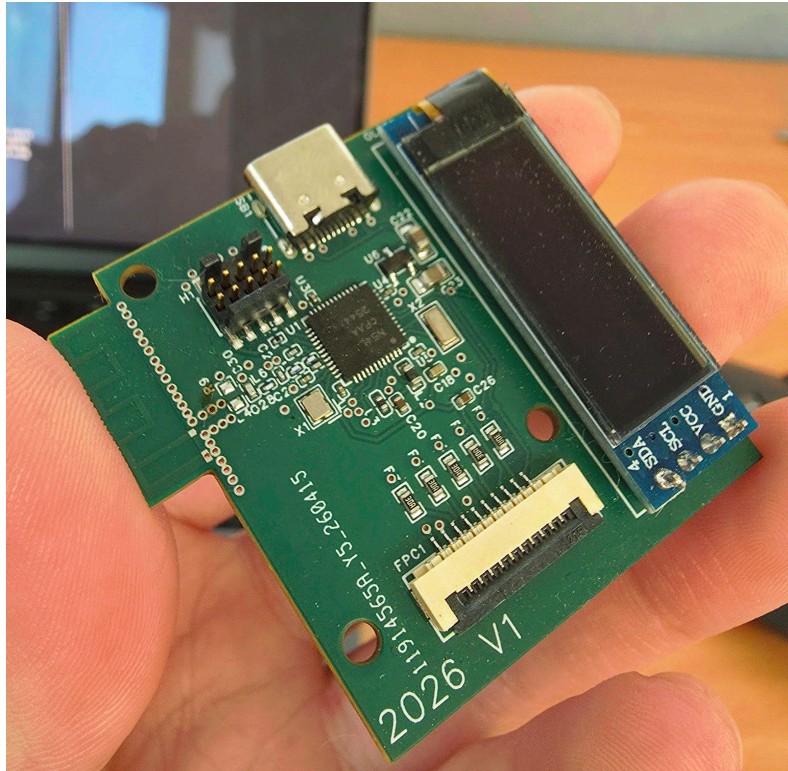
또한 본 제품 개발인원이 항상 SWD interface를 통하여 BLE 인증번호를 확인해서 페어링할 수도 없고 매점에 상주하여 해당기기를 관리할 수 없음으로 매점 상주 인원이 쉽게 페어링, 상태확인, 트러블슈팅 및 개발팀을 위한 오류보고를 할 수 있게 보드에 OLED 모듈을 하나 추가하여 상기한 내용이 표시될 수 있도록 구성하였다.



전원은 보드 상단의 USB C 포트를 통하여 인가받으며, CC핀에 5.1k 저항(뒷면)을 달아 USB Sink 5V 전압을 VBUS 라인에 받는다. 해당 전압은 LDO를 통하여 3.3V로 전환되어 VDD라인에 공급되며 LDO 전후단에 디커플링을 추가하여 공통 전원라인의 안정성을 높였다.



2.4GHz 안테나는 간단하게 해결하기 위해 TI AN043 Application note⁵ 를 보고 IFA를 서킷에 그대로 옮긴 후, Nordic semiconductor의 Q&A를 통하여 설계를 검증받았다. Feed 트레이스는 Digikey 트레이스 임피던스 계산기⁶를 이용하여 파라미터를 정해주었다 다음은 PCBA 주문을 통하여 받은 보드이다.



3.5. 트러블슈팅

보드를 받은 이후에 전원을 연결한 이후 DAP에 붙어보려고 시도했으나 Failed to Power Up DAP 오류가 발생하였다. 리턴받은(읽은) 값이 랜덤성의 띄었고 정상적으로 결과값을 돌려주지 않고 있다는 것 또한 확인하였다. DAP 자체가 응답을 주지 못한 것으로 판단하였다.

```
TF9FD58C0 005:619.607 Device "NRF54L15_M33" selected.
TF9FD58C0 005:620.746 - 2.079ms returns 0x00
TF9FD58C0 005:620.772 JLINK_EnableLog(...)
TF9FD58C0 005:620.779 - 0.008ms
TF9FD58C0 005:620.993 JLINK_GetEmuCaps()
TF9FD58C0 005:621.020 - 0.026ms returns 0xB8EA5A33
TF9FD58C0 005:621.035 JLINK_TIF_GetAvailable(...)
TF9FD58C0 005:621.214 - 0.178ms
TF9FD58C0 005:621.234 JLINK_TIF_Select(JLINKARM_TIF_SWD)
TF9FD58C0 005:621.884 - 0.650ms returns 0x00
TF9FD58C0 005:621.901 JLINK_IsConnected()
TF9FD58C0 005:621.909 - 0.008ms returns FALSE
TF9FD58C0 005:621.917 JLINK_SetSpeed(5000)
TF9FD58C0 005:622.061 - 0.144ms
TF9FD58C0 005:622.070 JLINK_Connect()
TF9FD58C0 005:622.560 ConfigTargetSettings() start
TF9FD58C0 005:622.570 J-Link Script File: Executing ConfigTargetSettings()
TF9FD58C0 005:622.590 ConfigTargetSettings() end - Took 17us
TF9FD58C0 005:622.599 InitTarget() start
TF9FD58C0 005:622.605 J-Link Script File: Executing InitTarget()
TF9FD58C0 005:625.792 InitTarget() end - Took 3.18ms
TF9FD58C0 005:626.736 Found SW-DP with ID 0x6BA02477
```

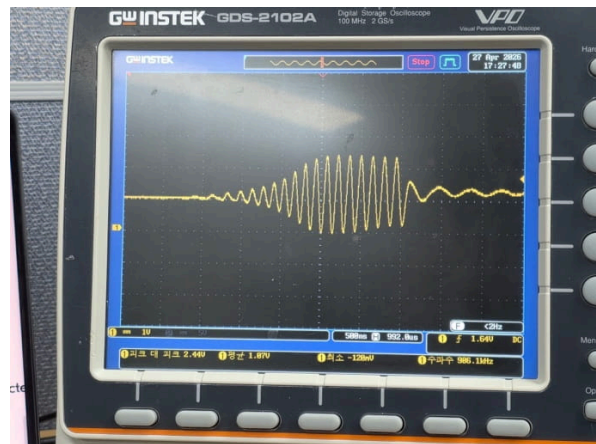
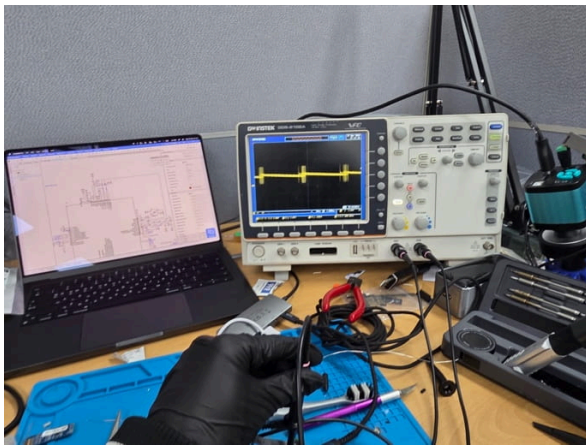
```

TF9FD58C0 005:628.365 Failed to power up DAP
TF9FD58C0 005:680.817 ConfigTargetSettings() start
TF9FD58C0 005:680.940 J-Link Script File: Executing ConfigTargetSettings()
TF9FD58C0 005:680.953 ConfigTargetSettings() end - Took 9us
TF9FD58C0 005:681.040 InitTarget() start
TF9FD58C0 005:681.055 J-Link Script File: Executing InitTarget()
TF9FD58C0 005:682.630 InitTarget() end - Took 1.53ms
TF9FD58C0 005:788.991 Failed to attach to CPU. Trying connect under reset.
TF9FD58C0 005:790.139 Found SW-DP with ID 0x6BA02477
TF9FD58C0 005:790.720 SWD speed too high. Reduced from 4000 kHz to 2700 kHz for stability
TF9FD58C0 005:791.796 Failed to power up DAP
TF9FD58C0 005:791.837 - 169.767ms returns 0xFFFFFEFB
TF9FD58C0 011:529.226 JLINK_IsOpen()
TF9FD58C0 011:529.294 - 0.068ms returns 0x01
TF9FD58C0 011:534.575 JLINK_Close()
TF9FD58C0 011:543.752 - 9.176ms
TF9FD58C0 011:543.810
TF9FD58C0 011:543.820 Closed

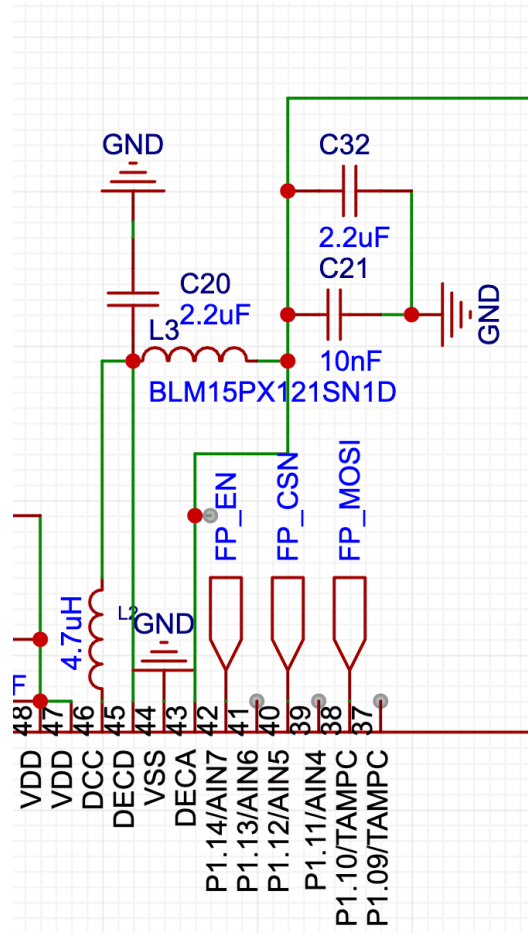
```

가장 먼저 의심했던건 당연히 오류에 나왔던 Power라인이었다. 10번핀에 매핑한 100nf 디커플링에 3.3v 전원라인을 실수로 연결을 해주지 않은 것을 발견하였는데 해당 핀을 오실로스코프로 찍어봐도 전압은 안정적이었고, 전원라인을 연결해줘도 다르지 않았다.

해당 부분이 원인이 아니라고 판단하고, 다른 위치에 프로브를 찍어가며 디버깅을 이어가던 중, DECA 라인이 발진하는 것을 발견하였다.



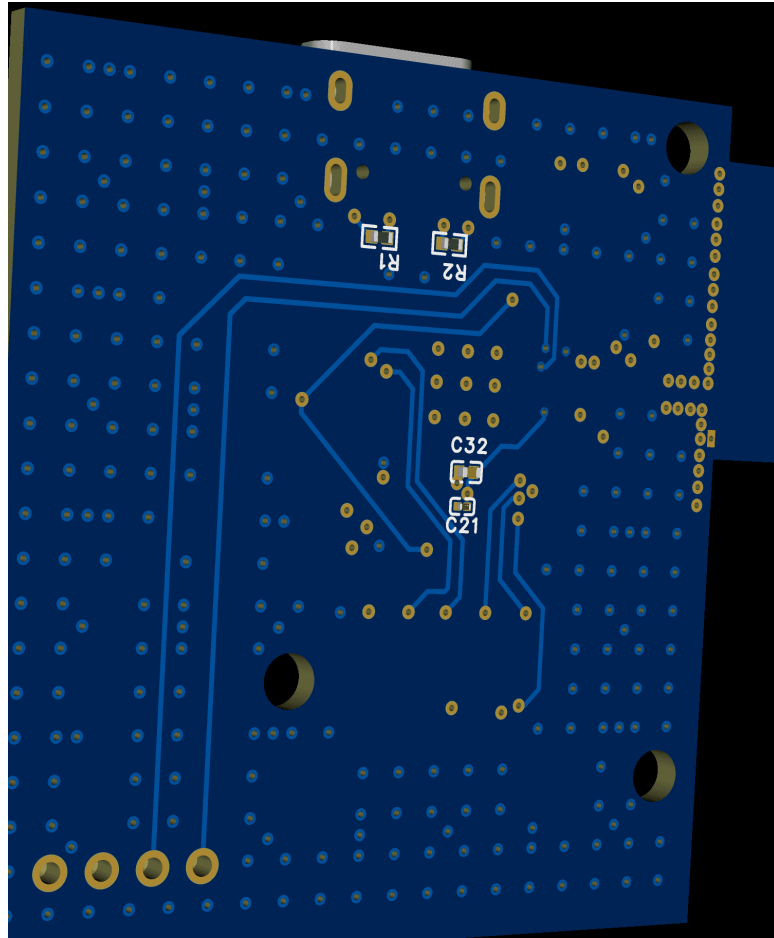
이후 여러 부품들을 리솔더링 해가며 원인규명을 시작하였는데,



위 회로의 C21을 제거하니까 발진이 사라지는 것을 발견하였다. (C32는 원인규명,문제해결 이후 생산에서 문제를 제거하기 위하여 추가한 컴포넌트다. PCB에는 존재하지 않는다.). 해당 현상을 확인 후 고민을 거듭한 이후, LC회로로 인한 발진보다는 (주파수가 8MHz였기에 어렵다고 보았다.) 벡컨버터 출력이 c21에 의해 아주 조금 스무딩 된거라고 생각하였다.

C21을 제거하고 발진이 사라진 이유로는 nRF54L15는 내부 전원단에 LDO와 벡컨버터가 있는데 스타트업할때는 LDO로 전원을 받으면서 벡컨버터 출력단에 LC회로를 감지하고 감지하지 않으면 LDO로 계속 동작하는 과정에서 **모종의 이유로 인덕터 감지에 실패하여 해당 현상이 발생한 것이 아닐까**라는 생각을 하였다.

해당 판단의 근거로는 인덕터 감지회로는 인덕터 자체라기보다는 LC회로가 발생시키는 발진을 감지하는 것에 가까운데, 10nF커패시터가 애매하게 보드 뒷면에 있어 감지가 된 상태에서 벡컨버터가 동작하여 해당 주파수로 발진이 일어나는 것인가 하는 생각을 하였다. 그리하여 ref circuitary와 내 회로를 비교해가며 확인해본 결과 2.2uf(C31)이 빠진 것을 확인하고 추가한 이후 문제가 해결되었다. 이후 Nordic devzone⁷에 올린 답변에도 C31이 빠졌다는 답변을 받아 C31의 문제를 확인받았다.



뒤에붙은 C21에 모습

하지만 문제 해결이후 해당 문제를 돌아보며 해당 판단에는 문제가 있었음을 발견했는데, 벡컨버터 스위치 출력은 DCC핀이고 해당 핀에는 이미 커패시터와 인덕터가 연결되어있다. 아래 다이어그램⁸에서 해당 내용을 확인할 수 있다.

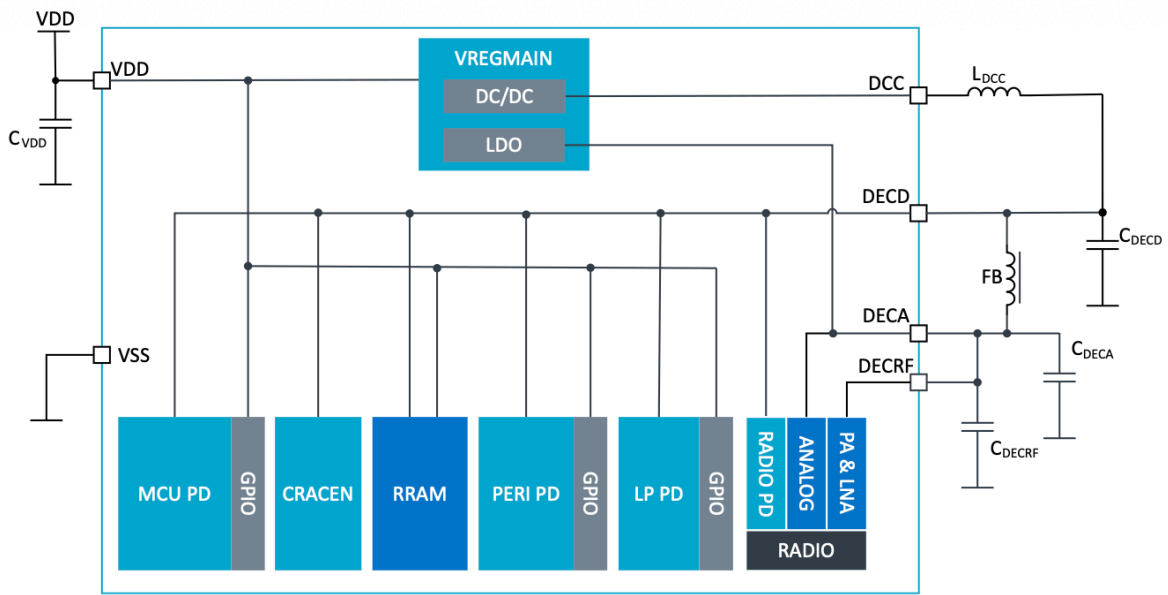


Figure 21: Regulator configuration

The main supply voltage is connected to the **VDD** pin.

DECA에서 볼 수 있듯이, DECA는 LDO에 직결된 라인이며 해당 라인의 디커플링이기에 스타트업 과정이나 아날로그 Peripheral 초기화 단계에서 문제가 발생했음을 알 수 있다. (원인규명 단계에서 **DECRF 연결부 트레이스를 끊고** 테스트를 해봤을때 **DECRF 라인은 안정적이었으며 DECA라인은 여전히 불안정했다.** 즉 RF쪽으로 인한 문제를 배제하였다.)

여기서 문제가 발생한다. 전원 안정화 실패가 원인이라면 10nf(C21)을 제거했을때 동작하지 않는 것이 이상적이다.

이후 이를 의문으로 남겨둔 이후 다른 프로젝트를 진행하다 문득 “**LDO에 디커플링 커패시터가 왜 있어야 하는가**”에 대한 의문이 발생하여 해당 사항에 대해서 조금 파고들어보다가 LDO는 **피드백 루프를 이용하여 내부 P MOS 게이트를 조절하여 저항을 만들어내 분압을 통한 타깃 전압을 만들어내는 구조임**을 알았다. (기존에는 LDO는 수동적인 소자로 생각하고 여기에 의문을 품지 않았다.) 이를 통해서 LDO 출력단 커패시터는 **부하의 증감으로인하여 LDO의 피드백루프가 적용되기 전, 저항값이 변하고있을때의 전압 상승/하강을 억제하기 위한 소자임** 또한 알 수 있었다.

이후 해당 깨달음을 바탕으로 10nF 커패시터가 **출력 전압의 변화와 피드백 루프로 인하여 변하는 저항값의 변화가 반전되어 발진을 일으켰고**, 이를 억제해줄 2.2uF 커패시터가 부재하여 나타난 현상임을 알 수 있었다.

3.6. 인프라 설계

3.6.1. DB 분리

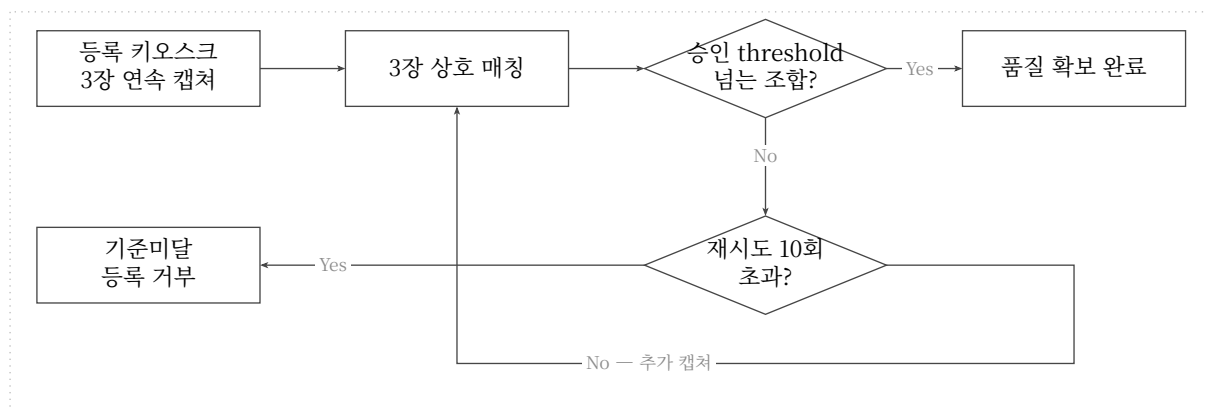
지문 DB는 개인정보보호위원회의 생체정보 보호 가이드라인에 따라, **Dimipay 서비스의 다른 DB와 다르게 분리된 다른 독립 DB로 구성되며**, AWS RDB를 이용한다. DB에 저장되는 정보는 **암호화되며**, DB의 인바운드 트래픽은 **디미페이 서버 IP로 제한이 걸리고 아웃바운드는 허용되지 않는다**. DB 패스워드는 랜덤 생성된 19자의 영대소문자, 숫자, 특수문자 혼합의 문자열을 사용한다.

3.6.2. 지문 서버 보안

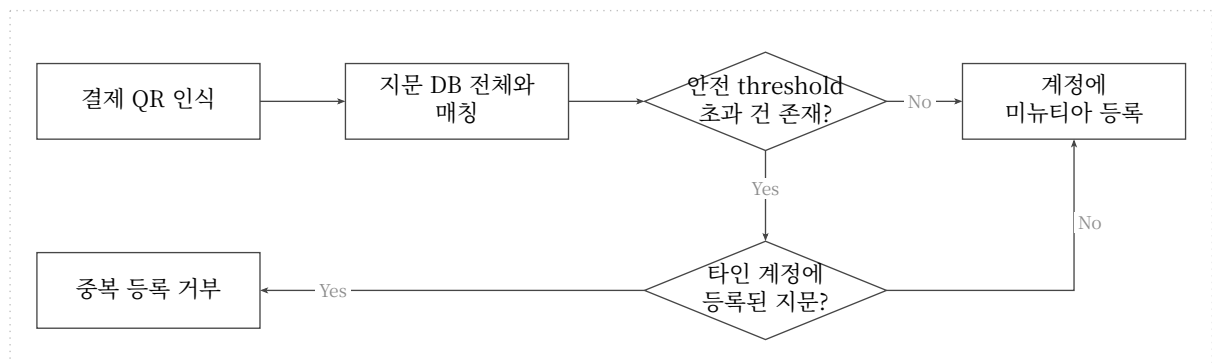
지문 DB와 직접 커넥션을 이루는 SourceAFIS JAVA 웹서버는 Version 1에서의 구성과 동일하다. **내부망에만 올라와있으며**, 혹시 모를 SSRF 공격에 대비하여 내부망에서도 **토큰 인증을 이용**한다.

3.7. 알고리즘 보완

등록 이미지 품질 검증



사용자는 기본적으로 별도 등록 키오스크를 이용하여 등록을 수행한다. 최소 3장의 지문 이미지를 BT_COMMAND_FINGERPRINT_RQ_DETAIL로 캡처하며, **각 지문 이미지끼리 매칭을 진행하여 score가 지문 승인 threshold를 넘지 못하면 추가 캡처를 받는다**. 결론적으로 3개의 이미지로 상호 매칭을 진행하였을때 지문 승인 threshold를 넘는 조합이 발생할때까지 반복되며, 이같은 과정이 10번 이상 반복되면 기준미달로 등록을 거부한다.



퀄리티 높은 지문 이미지를 확보한 이후에는 사용자의 결제 QR을 인식받아 계정에 등록한다. 다만 등록시 지문 DB 전체에 대하여 등록될 지문을 매칭한다. 이후 안전 threshold 점수를 넘지만 해당 지문이 다른 사용자의 계정에 등록되어있다면 등록을 거부한다. (타인 계정에 대하여 지문등록 거부)

또한 한 유저는 같은 name을 가진 여러 미뉴티아 레코드 묶음을 다수 보유할 수 있으며, 이를 통하여 한 지문을 같은 이름으로 여러번 등록하여 FRR을 완화할 수 있고 다른 손가락은 분리하여 등록할 수 있다.

앞 단계는 본인의 지문이 충분히 좋은 품질로 찍혔는가 를 보고, 뒷 단계는 그 지문이 이미 다른 사람의 것으로 등록되어 있지 않은가 를 본다. FRR을 잡으면서 여러 계정에 같은 지문이 등록됨에 따라 발생하는 예외를 잡기 위함이다.

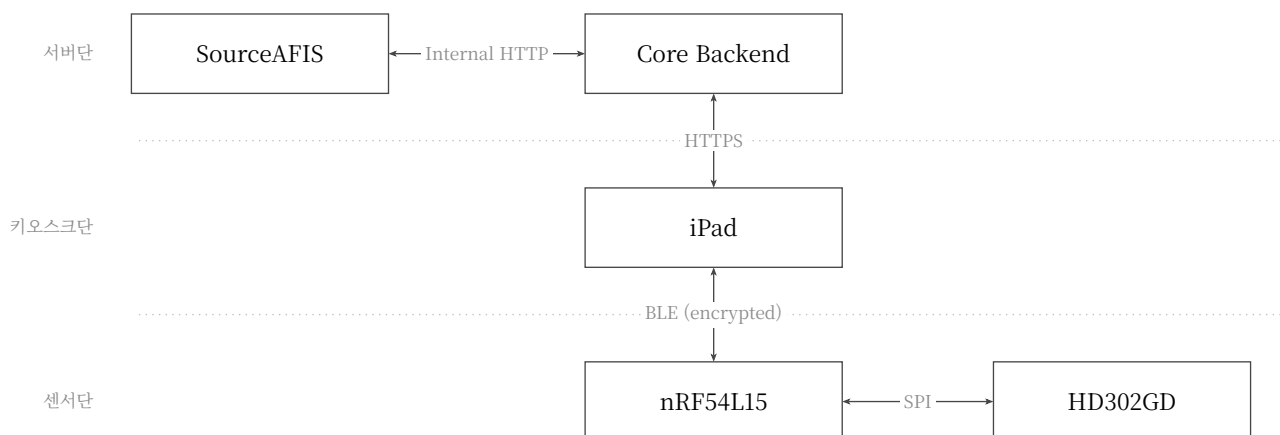
3.7.1. 지문삭제

앱에서 지문 그룹을 관리 가능하다. 삭제/그룹이름변경 등이 가능하다. 또한 유저가 더이상 지문 인식결제를 이용하고 싶지 않아 앱에서 삭제버튼을 클릭함으로써 요청하거나 이용자가 졸업하는 경우 해당 개인정보가 불필요해졌음으로 개인정보 보호법 제 21조에 의거하여 운영 DB에서 즉시 삭제한다. 다만 시스템 장애 복구를 위한 백업본에는 백업 순환 주기에 따라 해당 정보가 최대 3일간 잔존할 수 있으며, 이 기간 동안 해당 정보는 서비스에서 접근하거나 처리할 수 없다. 백업본에 잔존한 정보 역시 백업 보존기간이 만료되는 즉시 영구적으로 파기한다.

3.8. 펌웨어 설계

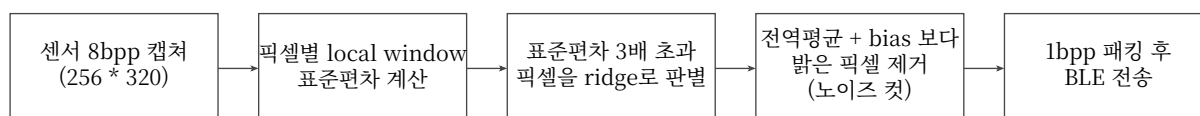
3.8.1. 지문센서 통신

지문센서 데이터시트⁹ 레지스터테이블에 자세한 설명이 달려있지 않고 레지스터 이름밖에 없어, 제조사에서 제공한 AS608칩이 이용된 **테스트 보드의 소스¹⁰**를 보고 프로토콜을 유추/복구하여 펌웨어를 설계하였다. 초반 테스트보드 소스 없이 구현할때 레지스터 테이블에 각 레지스터별 정확한 설명이나 라이프사이클이 명시되어있지 않아서 개발에 어려움이 있었다.



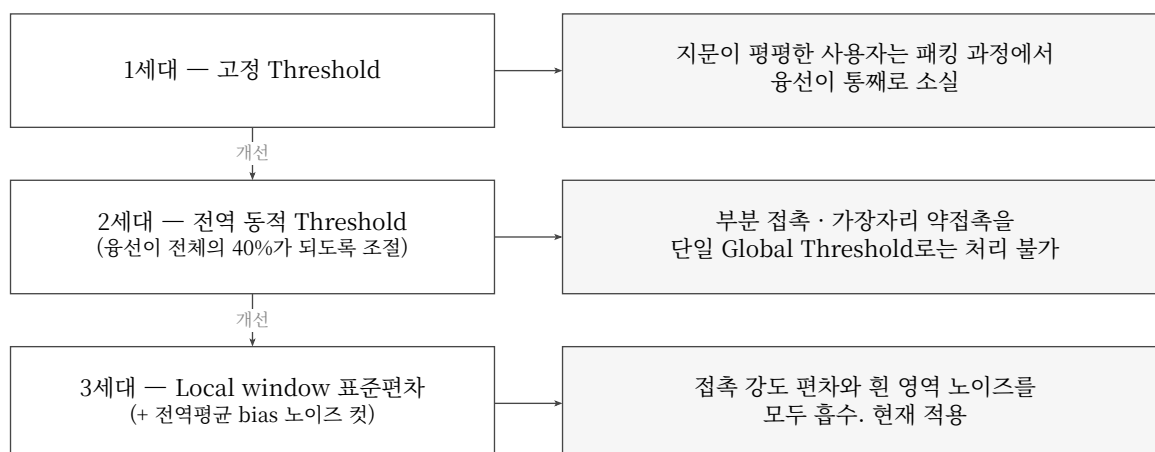
전체적인 통신 구조는 위와 같다. V1에서 바뀐 것은 크게 없다.

3.8.2. 이미지 처리



이미지 데이터는 **260*320 1bpp** 이미지를 제공한다. 원래 센서 자체는 8bpp 이미지를 제공하나 전송속도의 개선을 위하여 1bpp로 변환하여 전송한다. 최종적으로 적용된 변환 파이프라인은 다음과 같다.

이 파이프라인은 세 번에 걸쳐 개선된 결과이다.

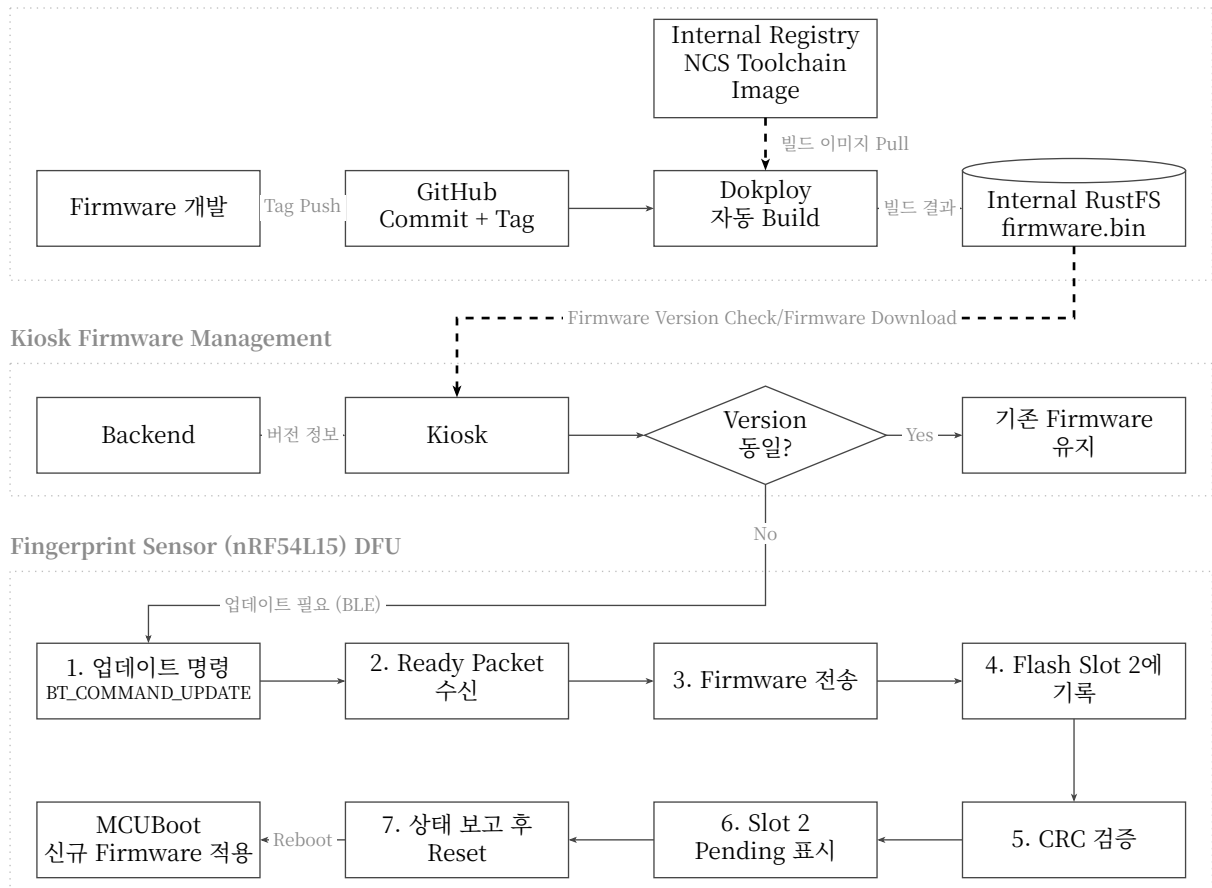


1bpp 패킹시에 처음에는 **고정 Threshold**를 이용하다가 지문 용선의 생김새가 사람마다 다른 관계로 **평평한 지문을 가진 사람들은 패킹시 용선이 소실되는 문제가 존재하였다.**

이를 인지하고 융선이 전체 이미지의 40%를 차지하게 Threshold를 유동적으로 조절하는 알고리즘을 적용하였으나 손가락을 완전히 접촉시키지 않아 이미지 프레임 안에 지문이 찍차지 않는 사용자가 존재하였고 지문 인식시 주로 가운데 부분은 강하게 접촉되고 테두리부분이 약하게 접촉되에 하나의 Global Threshold를 이용하는 방식에 한계를 확인하였다. 이후 각 픽셀에 대한 local window를 만들어서 그 window 안에서 표준편차를 구하고 표준편차 3배 이상의 차이가 나는 픽셀을 융선(검은 영역)으로 판별하는 알고리즘을 작성하였다. 하지만 이런 구현에서 지문이 닿지 않은 흰 영역에서 노이즈가 융선으로 판별되는 경우가 발생하였고 전역 평균의 + bias 보다 밝은 픽셀은 죽여 노이즈를 제거하였다.

3.8.3. DFU (Device Firmware Update)

Firmware Build / Deploy



이는 유지보수 과정에서 추가된 기능인데 펌웨어 업데이트마다 분해해서 Jlink로 업데이트 하는 행위가 상당히 비효율적인 관계로 블루투스를 통해서 키오스크로 일괄 업데이트가 가능하게 구성하였다. 펌웨어를 작성하여 커밋에 태그를 붙여 깃허브에 커밋하면 Dokploy가 빌드 이후 .bin 파일을 내부 RustFS에 배포한다. 키오스크는 일정 주기로 지문센서와 백엔드로부터 중계받은 활성 펌웨어 버전을 비교하며, 버전이 다른 경우 백엔드로부터 펌웨어를 중계받아 DFU를 수행한다.

키오스크가 수신한 펌웨어는 다음과 같은 과정으로 지문센서에 적용된다.

1. 키오스크가 BT_COMMAND_UPDATE를 호출한다.
2. 지문센서가 Ready 패킷을 송신한다.
3. 키오스크가 펌웨어 파일을 지문센서로 전송한다
4. 지문센서는 Flash slot 2로 펌웨어 파일을 스트리밍한다.
5. 지문 수신이 완료된 후 CRC 검증을 수행한다
6. Slot 2를 pending으로 표시한다.
7. 상태보고 후 Reset을 수행한다.

nRF54L15 펌웨어를 빌드하기 위해서는 NCS(nRF Connect SDK)가 필요한데 이 환경 구성을 매 빌드에서 하면 너무 길어짐으로 해당 환경이 세팅된 도커 이미지를 내부 Registry에 올려두고 해당 이미지를 이용하였다.

키오스크에서 실제로 DFU가 수행되는 과정은 [시연 영상](#)에서 확인할 수 있다.

3.9. 성능



V1의 평균 인증 지연시간 5.8초를 V2에서 **1.38초로 단축**하여 약 **76.2%의 지연시간 감소**를 달성했다. 가장 큰 병목이었던 센서→MCU 전송은 UART 기반 3.8초에서 SPI 기반 360ms로 약 **90.5% 감소**하였다. 이로써 초기 요구조건인 **2초 이내 인증**을 만족하였다.

4. 성과

완성된 시스템은 실제 교내 매점에 배포되어 운영되었다. 매장에서의 실제 결제 동작은 [시연 영상¹²](#)에서 확인할 수 있다.

```
.service.ts  kiosk.service.impl.ts  fingerprint  endpoints.ts  fingerprint.controller.ts  console
Tx: 자동  Playground
1 SELECT * FROM transaction WHERE created_at BETWEEN '2026-05-01' AND '2026-05-31';
2
3 ✓ SELECT
4     count(*) AS "success",
5     count(*) FILTER (WHERE transaction_type = 'FINGERPRINT') AS "fingerprint",
6     count(*) FILTER (WHERE transaction_type = 'FINGERPRINT')::numeric / count(*) * 100 AS "ratio",
7     sum(total_price) FILTER (WHERE transaction_type = 'FINGERPRINT')
8 FROM transaction
9 WHERE created_at >= DATE '2026-05-01'
10    AND created_at <= DATE '2026-05-31'
11    AND status = 'CONFIRMED';
```

	success	fingerprint	ratio	sum
1	7454	3199	42.916554869868526965	5820610

2026년 5월 한달동안 있던 7454건의 성공적인 결제에서 3199건의($\approx 43\%$) 결제가 지문인식 결제로 진행되며 순수 지문인식 결제로 582만원의 월간 결제액을 기록하며 높은 사용률을 보였다.

5. 참조자료

¹ https://github.com/yeonfish6040/zkfinger_sdk

² https://youtu.be/aYEx_O_Kxps

³ https://youtu.be/aYEx_O_Kxps

⁴ https://docs.nordicsemi.com/bundle/ps_nrf54L15/page/chapters/ref_circuitry.html

⁵ <https://www.ti.com/lit/an/swra117d/swra117d.pdf>

⁶ <https://www.digikey.kr/ko/resources/conversion-calculators/conversion-calculator-pcb-trace-impedance>

⁷ <https://devzone.nordicsemi.com/f/nordic-q-a/127910/nRF54L15-cannot-use-swd-interface>

⁸ https://docs.nordicsemi.com/r/bundle/ps_nrf54l15/page/regulators.html

⁹ https://www.adh-tech.com.tw/uploads/1764979488_HF302GD%20Module%20datasheet.pdf

¹⁰ https://www.adh-tech.com.tw/uploads/1764979770_HF302GD_AS608_V2.3.4.x.zip

¹¹ <https://www.youtube.com/watch?v=tb2I9RlrJJ4&t=17s>

¹² <https://www.youtube.com/watch?v=HWvwYAcl2mI&t=73>