
Chapter 214. NFC

1. RF (Radio Frequency)

RF(Radio Frequency)는 무엇일까? ‘라디오’라고 하면 흔히 차를 타고가면서 들을 수 있는 라디오 방송이 제일 먼저 떠오르고, ‘라디오 주파수’라고 하면 라디오 방송에 사용하는 주파수를 생각하게 된다. 하지만 우리가 흔히 듣는 ‘라디오 방송’은 실제 ‘라디오’를 이용하는 한 가지 예에 불과하며 ‘라디오 방송’이 가장 흔히 접할 수 있는 ‘라디오’를 이용한 예이기 때문에 고착화되었을 따름이다. 사전에서 라디오의 정의를 찾아보면 라디오 방송이나 라디오를 청취할 수 있는 기기가 가장 먼저 나오지만 보다 일반적인 ‘무선’ 또는 ‘무선 통신’이라는 뜻 역시 볼 수 있으며 이것이 바로 여기서 다루고자 하는 ‘라디오’의 의미이다.

RF(무선 주파수)는 기술적으로 주어진 주파수의 신호를 안테나로 흘려보냈을 때 무선 방송이나 통신에 적합한 전자기장이 생성되는 특성을 갖는 교류를 지칭한다. 이러한 주파수에는 사람이 들을 수 있는 가청 주파수 대역에 포함되는 9KHz에서부터 수천 GHz에 이르기까지 넓은 영역이 포함된다. 무선 주파수 영역은 주파수에 따라 표 1과 같이 몇 개의 대역으로 구분할 수 있다.

명칭	약어	주파수	파장	응용
초장파 (Very Low Frequency)	VLF	9KHz~30KHz	33Km~10Km	선박
장파 (Low Frequency)	LF	30KHz~300KHz	10Km~1Km	항해
중파 (Medium Frequency)	MF	300KHz~3MHz	1Km~100m	AM, 항공
단파 (High Frequency)	HF	3MHz~30MHz	100m~10m	아마추어 무선, 단파 방송
초단파 (Very High Frequency)	VHF	30MHz~300MHz	10m~1m	FM, TV
극초단파 (Ultra High Frequency)	UHF	300MHz~3GHz	1m~100mm	이동전화
초고주파 (Super High Frequency)	SHF	3GHz~30GHz	100mm~10mm	위성
마이크로파 (Extremely High Frequency)	EHF	30GHz~300GHz	10mm~1mm	우주통신

표 1. 무선 주파수 대역

RF는 현재도 수많은 분야에서 사용되고 있으며 그 범위도 계속 넓어지고 있다. RF라는 용어가 무선 주파수를 의미하지만, 무선 주파수에 의해 만들어지는 전자기파를 이용하는 장치 및 그 기술들까지도 포함하는 의미로 보다 일반적으로 사용되고 있다. 또한 표 1의 주파수 대역 중에서도 일반적으로 수백 MHz에서 수 GHz 까지의 고주파 영역이 RF 영역으로 인식되고 있다.

RF와 구별해야 하는 것에는 무선 통신이 있다. RF는 ‘특정 주파수에 의해 만들어지는 전자기파를 이용하는 장치 및 기술’을 의미하므로 무선 통신의 측면에서 보자면 실제로 신호를 보내고 받는 부분이 RF 분야에 속하게 된다. 무선 통신을 위해서는 전파의 송수신 이외에도 수많은 분야가 포함되어 있지만 무선 통신에서 RF는 신호 전달을 위한 핵심적인 부분을 차지하고 있다. 반면 RF 측면에서 보자면 무선 통신은 RF를 이용하는 응용 분야의 한 가지로 볼 수 있다. 아두이노에서 사용할 수 있는 무선 통신 기술인 WiFi, 블루투스, 지그비, NFC, RFID 등은 모두 RF를 사용하고 있다. 우리가 접할 수 있는 RF 사용 예는 대부분 무선 통신 분야이지

만 이외에도 측정, 검사, 탐사, 보안 등 다양한 분야에서 사용되고 있다.

분야	예
군용	GPS, 위치 추적 스마트 무기
천문학 및 우주 탐험용	우주 공간 탐사
통신용	HDTV, 컴퓨터 네트워크, 이동 전화
의학용	심장 자극기, 살균 소독, CT, X-ray
원격 탐사용	기상 관측, 천연 자원 탐사, 수중 탐사
감시용	무인 경비 시스템, 도청 탐지기
기타	충돌 방지, 구조 정밀 검사, 버스 카드, 주차 관리 시스템

표 2. RF 응용 분야

RF의 대표적인 응용 중 하나에 RFID(Radio Frequency Identification)가 있다. RFID란 데이터가 기록된 IC 칩과 RF를 사용하여 다양한 정보를 인식하고 관리할 수 있는 기술을 말한다. RFID는 전자 태그, 스마트 태그, 전자 라벨, 무선식별 등으로도 불린다. RFID는 여러 개의 정보를 동시에 판독하거나 수정 및 갱신이 가능하다는 장점으로 바코드 기술이 극복할 수 없는 여러 가지 문제점들을 해결할 수 있어 물류 관리나 보안 등의 분야에서 널리 쓰이고 있다. 표 3은 바코드와 RFID를 비교한 것으로 태그를 위한 비용을 제외하면 모든 면에서 RFID는 바코드에 비해 우수한 것을 알 수 있다. 하지만 RFID의 태그 역시 가격이 계속해서 하락하고 있는 추세에 있다.

항목	바코드	RFID
인식 방법	비접촉식	비접촉식
인식 거리	~50cm	~27m
손상률	잦음	거의 없음
데이터	~100Byte	~64KByte
보안	거의 없음	복제 불가능
재활용	불가능	가능
비용	낮음	상대적으로 높음

표 3. 바코드와 RFID 비교

주변에서 흔히 볼 수 있는 RFID 사용 예에는 버스에서 사용하는 교통카드, 고속도로의 하이패스 카드, 마트에서 도난 방지용으로 사용하는 태그 등이 있다. RFID를 사용하기 위해서는 태그와 태그로부터 정보를 읽어 들일 수 있는 판독기가 필요하다. 태그는 정보가 기록되어 있는 IC 칩과, 칩에 저장된 정보를 판독기로 전송할 수 있는 안테나로 이루어져 있다.



그림 1. RFID 태그

RFID는 사용하는 전원에 따라 몇 가지로 분류된다. 판독기의 전원만으로 동작하는 RFID를 수동형(passive) RFID라고 한다. 수동형 RFID의 경우 판독기의 안테나에서 만들어지는 전자기장으로부터 상호 유도 방식으로 태그의 IC 칩을 동작시킬 수 있는 전력을 만들어 사용한다. 인식 거리가 짧고 판독기에서 많은 전력을 소비하는 단점이 있지만 태그를 작고 간단하게 만들 수 있어 많이 사용되고 있다. 반면 태그에 건전지가 내장되어 있어 칩의 정보를 읽기 위해서는 내장 전원을 사용하고 통신을 위해서는 판독기의 전원을 사용하는 RFID를 반수동형(semi-passive) RFID라고 한다. 반수동 RFID는 판독기로부터 신호를 받을 때까지 작동하지 않는 상태로 있으므로 전력 소비가 적어 오랜 시간 사용할 수 있다. 마지막으로 능동형(active) RFID는 태그의 정보를 읽고 정보 송신을 위해 태그의 전원을 사용하므로 수십 미터에 달하는 원거리 통신용으로 사용된다.

RFID는 통신에 사용하는 주파수에 따라서도 몇 가지로 분류된다. LFID(Low Frequency Identification)는 120~140KHz 주파수를 사용하는 RFID이며, HFID(High Frequency Identification)는 13.56MHz 주파수를 사용한다. UHFID(Ultra High Frequency Identification)는 가장 높은 969~956MHz 주파수를 사용한다.

HFID는 RFID 중 가장 널리 사용되는 방식으로 수동형 태그를 사용하고 60cm 이내의 짧은 거리에서 태그의 정보를 읽고 쓸 수 있어 물품 관리, 교통카드, 출입/보안 카드 등에 흔히 사

용되고 있다. HFID는 NFC와 동일한 주파수를 사용하므로 NFC와 호환이 가능하다. 아두이노용 확장 설드로 판매되는 제품들은 RFID(정확하게 이야기하자면 13.56MHz를 사용하는 HFID)와 NFC를 동시에 지원하는 경우가 대부분이다. 그림 2는 13.56MHz를 사용하는 수동형 RFID의 구조와 동작을 나타낸 것이다.

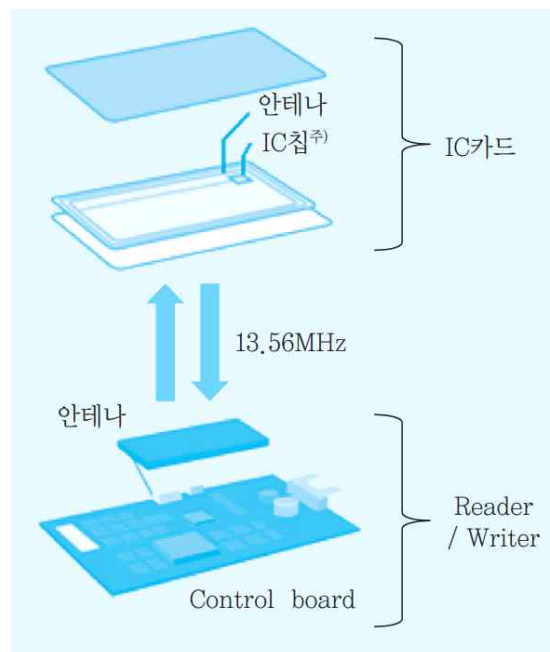


그림 2. RFID 동작

2. Near Field Communication (NFC, 근거리 무선 통신)

스마트폰의 이용은 음성 통화 중심에서 데이터 통신 서비스(SMS (Short Message Service), E-Mail, 웹 서핑 등)를 거쳐 생활 편의 서비스로 진화하고 있는 중이다. 이러한 편의 서비스를 가능하게 하는 기술 중 하나가 NFC로 최근 모바일 산업에서 주목을 받고 있다.

NFC는 1999년 특허가 등록된 이후 2002년에 Sony와 NXP가 기존 RFID 기술을 확장하여 만든 개방형 RFID 기술의 일종으로, 13.56MHz 주파수 대역을 사용하여 10cm 이내의 가까운 거리에서 태그에 담겨진 정보를 단말기 사이에 전송하는 비접촉식 통신 기술을 말한다. RFID와 비슷하게 들리지 않는가? 사실 13.56MHz를 사용하는 RFID와 NFC는 통신 방식에서는 동일하지만 NFC에는 RFID와 몇 가지 면에서 차이가 있다. NFC의 가장 큰 특징은 짧은 통신 거리에 있다. NFC의 짧은 통신 거리는 단점이 될 수도 있지만, 사용자가 직접 터치하는 의도적인 행동이 필요하므로 행동 기반의 맞춤형 서비스들과 연계하기에 좋은 장점이 있다. NFC에는 보안 모듈이 탑재되어 있어 보안성이 높은 점도 장점 중 하나이다. 또한 NFC와 더불어

스마트폰에 포함되어 있는 블루투스의 경우 페어링(pairing) 과정을 거쳐야만 하는 번거로움이 있지만 NFC는 별도의 설정 없이도 사용할 수 있으며, 기존 근거리 무선 데이터 통신 기술이 읽기만 가능했던 반면, NFC는 읽기와 쓰기가 모두 가능하다는 등의 장점이 있다.

최근 스마트폰에도 NFC 기능이 담겨짐에 따라 일상생활의 많은 부분에서 응용이 가능할 것으로 예상된다. 예를 들어 NFC를 활용하여 스마트폰으로 출입문을 간편하게 열고 닫을 수 있으며, 버스카드 없이 스마트폰으로 대중교통을 손쉽게 이용할 수 있고, 스마트폰에 신용카드나 쿠폰을 저장해 쇼핑에 활용하는 것도 가능하다. NFC 태그를 부착한 포스터나 안내판에 스마트폰을 가까이 가져가면 관련 정보를 스마트폰을 통해 제공할 수 있으며, 종이 명함을 대신하여 스마트폰을 서로 접촉하는 것으로 명함 교환이 이루어지도록 할 수도 있다. 파일 전송 기능을 이용하여 스마트폰에 저장된 사진을 프린터로 전송하여 인쇄할 수 있으며, 친구와 전화번호 목록을 교환하는 것도 가능하다. 이렇듯 NFC는 RFID는 기반 기술로 하고 서비스 측면이 강조됨으로써 응용 가능한 분야는 무궁무진할 것으로 기대되고 있다.

NFC는 RFID와 마찬가지로 두 개의 장치 사이에서 하나는 판독기(reader/writer)로 다른 하나는 태그(tag)로 동작하여 반이중 방식(half-duplex)으로 정보를 주고받는다. 태그는 메모리와 안테나를 포함하고 있으며, 자체 전원을 포함하고 있지 않은 수동(passive) 장치의 경우 판독기가 태그 검출을 위해 만드는 전자기장(electromagnetic field)을 전력으로 변환하여 데이터를 전송한다. 판독기는 능동(active) 장치로 태그가 수동 장치인 경우 전자기장을 형성하여 태그가 동작할 수 있도록 하는 역할을 한다. 태그의 전원 유무에 따라 NFC 장치는 두 개의 통신 모드로 통신이 수행될 수 있다.

- 능동 모드 (active mode) : 능동 모드는 통신에 참여하는 두 장치 모두가 전원을 가지고 있는 경우로, 번갈아 신호를 전송함으로써 통신이 이루어진다.
- 수동 모드 (passive mode) : 수동 모드는 통신을 시작하는 시동기기(initiator)가 신호를 생성하고 목적기기(target)는 시동기기가 생성하는 전자기장을 이용하여 정보를 전송하는 모드이다.

위의 두 가지 통신 모드를 이용하여 NFC 장치는 세 가지 동작 모드를 가질 수 있다. 스마트폰의 예를 들어 설명해 보자.

- 읽기/쓰기 모드 (read/write mode) : 스마트폰이 판독기로 동작하는 모드로 태그에 데이터를 읽거나 쓸 수 있다. 이 때 스마트폰은 전원이 켜진 상태여야 한다.
- P2P 모드 (peer to peer mode) : 스마트폰은 다른 NFC 기기와 데이터를 교환할 수 있

다. 이를 위해서 스마트폰과 다른 NFC 기기는 모두 전원이 켜진 상태여야 한다.

- 카드 에뮬레이션 모드 (card emulation mode) : 스마트폰은 태그로 동작하며 다른 판독기가 스마트폰으로부터 데이터를 읽는다. 이 때 스마트폰은 전원 상태와 무관하게 태그로 동작할 수 있다.

이처럼 RFID는 태그와 판독기의 역할이 고정되어 있지만 NFC는 양방향 통신이 가능하므로 하나의 NFC 기기가 태그와 판독기의 역할을 모두 수행할 수 있다는 점에서 RFID와 차이점이 있다. 그림 3은 안드로이드를 운영체제로 사용하는 스마트폰에서 지원하고 있는 NFC의 기능에 대한 아이콘의 예를 나타낸 것으로, 능동 모드와 수동 모드로 구별되어 있다.

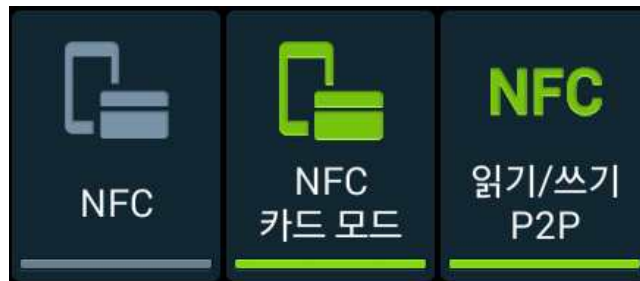


그림 3. 안드로이드 스마트폰의 NFC 아이콘

NFC 통신에 사용되는 13.56MHz 대역의 근거리 무선통신은 통신 범위에 따라 10cm 이내의 근접형(proximity)과 1m 이내의 주변형(vicinity)으로 나뉜다. 이 중 NFC를 포함하는 스마트카드에 적용되는 대표적인 표준에는 ISO/IEC 14443이 있다. ISO/IEC 14443에는 스마트카드의 크기 및 물리적 특성을 포함하여, RF 신호의 강도, 충돌 방지 및 데이터 전송 프로토콜을 정의하고 있다. ISO/IEC 14443 기반의 대표적인 칩은 NXP의 MiFare로 ISO/IEC 14443 Type A를 지원한다. 반면 1m 이내 범위에서의 무선 인식을 위한 표준으로는 ISO/IEC 15693이 있다. NFC는 2004년 처음으로 ISO/IEC 18092 표준으로 제정되었으며 ISO/IEC 18092에는 스마트 기기 사이의 통신을 정의하고 있어 기존 스마트카드와는 다른 부분이었다. ISO/IEC 18092에는 일본의 스마트카드 표준인 펠리카(FeliCa) 기술 일부가 반영되었다. 이러한 여러 가지 13.56MHz 대역의 무선통신 기술 표준들은 NFC의 확산을 위한 전략의 일환으로 NFC 범주에 모두 포함시킨 ISO/IEC 21481가 제정됨으로써 NFC는 현재와 같은 모습으로 정착되었다. 현재의 NFC는 ISO/IEC 14443을 기본으로 하고 태그와 판독기를 하나의 형태로 합쳐놓은 형태로 볼 수 있으며 기본적으로 스마트폰에서의 사용을 목적으로 만들어진 것이다.

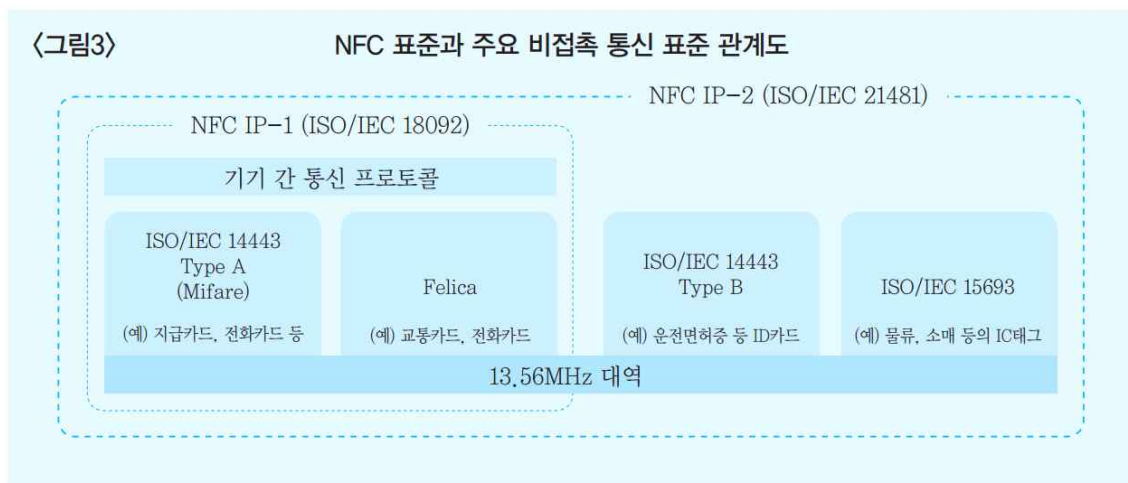


그림 4. NFC 표준 범위

ISO/IEC 14443은 다시 Type A와 Type B로 나뉜다. Type A는 13.56MHz 대역 비접촉식 카드에 대한 최초의 ISO 표준으로 세계적으로 가장 널리 사용되고 있다. Type A 카드는 흔히 메모리 카드(memory card)라고 불리는 반면, Type B 카드는 마이크로프로세서 카드(microprocessor card)라고 불린다. 소니에서 개발된 FeliCa는 ISO/IEC 14443의 Type C로 고안되었으나 표준으로 제정되지는 못하여 별도의 표준인 ISO/IEC 18092에 포함되었다. 각 Type들은 변조(modulation), 부호화(bit coding), 충돌방지(anti-collision) 등을 구현하는 세부 기술에서 약간의 차이가 있다. Type A의 전신으로도 불리는 MiFare는 Type A 표준을 따르면서 보안 프로토콜 등 일부 어플리케이션에 대해 자체 개발 모듈을 사용하는 형태로 구성되어 있다. Type A는 국내에서도 교통카드에 적용된 예가 있으며 일본에서는 전화카드에 사용되었다. Type B는 모토로라에서 개발한 표준으로 특정 특허에 종속되지 않아 주민등록증, 운전면허증과 같은 공공 용도나 각종 ID카드에 이용되고 있다. FeliCa는 스마트카드 일본 표준으로 일본 내에서는 교통카드, e-머니, 발권 시스템, ID카드 등 다양한 분야에 사용되고 있다.

NFC에서 사용할 수 있는 태그는 4가지 종류가 정의되어 있으며 통신 속도, 메모리 크기, 보안 기능 등 세부 내용에서 약간의 차이가 있다.

	Type 1	Type 2	Type 3	Type 4
표준	ISO 14443 A	ISO 14443 A	FeliCa (ISO 18092)	ISO 14443 A/B
속도 (Kbit/sec)	106		212, 424	106~424
메모리 (Byte)	96 ~ 2K	48 ~ 2K	≤ 1M	≤ 32K
충돌 방지 기능	×	○	○	○
응용 분야	저용량 태그		고용량 태그	

표 4. NFC 태그의 종류

NFC와 관련하여 마지막으로 언급하고 싶은 내용은 NDEF이다. NDEF는 NFC Data Exchange Format의 약어로 데이터 교환을 위한 표준 포맷에 해당한다. NDEF를 사용하지 않고도 데이터 교환은 가능하다.

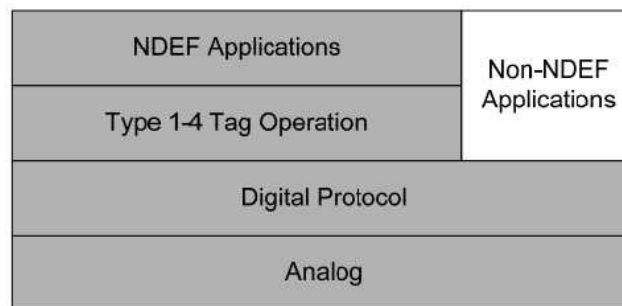


그림 5. NFC 동작 계층

NDEF는 간단한 이진 메시지 정의를 통해 어플리케이션에서 필요로 하는 페이로드(payload)를 하나의 구조에 넣을 수 있도록 만들어져 있다. NDEF 메시지는 그림 6에 나타난 것처럼 연속된 하나 이상의 NDEF 레코드로 구성된다.

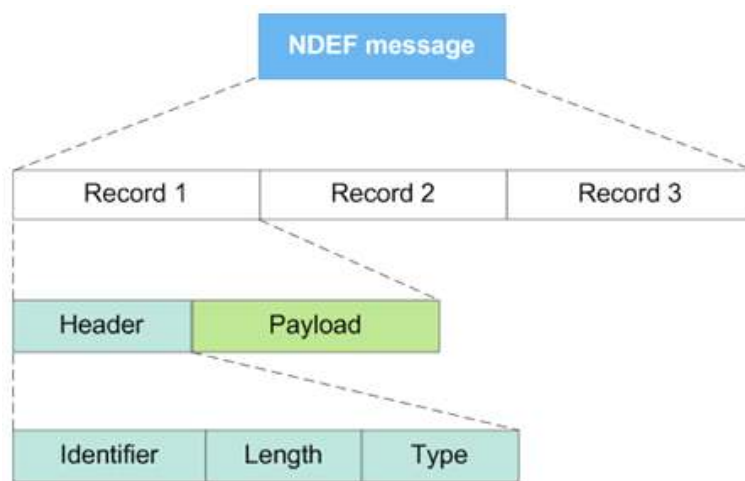


그림 6. NDEF 메시지 구조

이 책에서는 NDEF 메시지 구조에 대해 자세히 다루지는 않지만, 아두이노에서 NFC를 사용하기 위한 확장보드나 쉴드를 설명하는 문서에서 NDEF를 언급하는 경우가 있으므로 NDEF가 NFC를 통해 전달되는 정보 인코딩을 위한 표준 포맷이라는 점은 기억하도록 하자.

3. NFC 쉴드

이 장에서는 13.56MHz 주파수를 사용하는 RFID와 NFC 어플리케이션을 지원하는 PN532 칩을 사용하여 만들어진 쉴드를 사용한다.

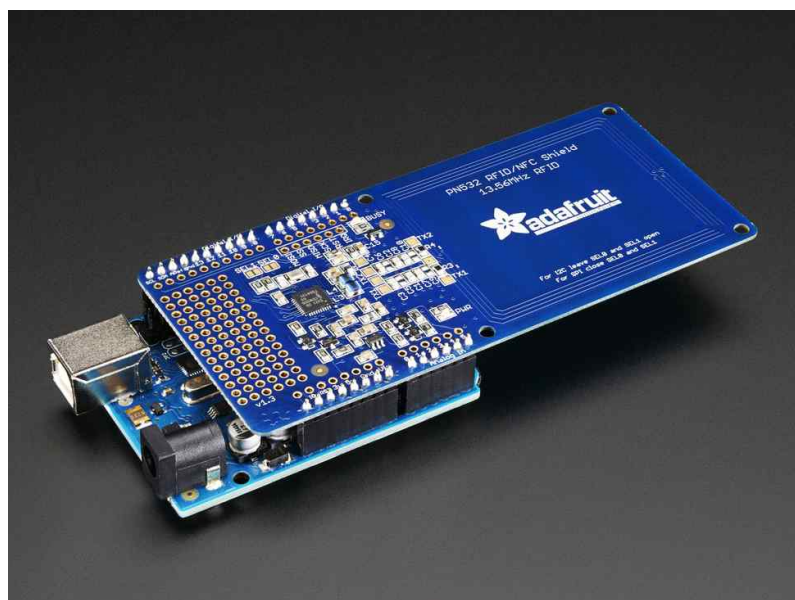


그림 7. NFC 쉘드¹⁾

PN532 칩은 NFC 기능을 지원하는 칩 중 가장 널리 사용되고 있는 칩으로 NFC의 모든 통신 모드를 지원하지만 그림 7의 쉘드는 읽기/쓰기 모드만을 지원하고 있다. NFC 쉘드는 디폴트로 I2C 통신을 사용한다. 또한 디지털 2번 핀을 인터럽트 핀으로 사용하고 있다. 인터럽트 핀은 쉘드에서 태그를 발견하였을 때 LOW 값을 가지므로 아두이노에서 지원하는 외부 인터럽트를 통해 폴링 방식이 아닌 인터럽트 방식의 처리가 가능하다.

NFC 쉘드를 사용하기 위해서는 먼저 전용 라이브러리가 필요하며 인터넷에서 무료로 다운로드할 수 있다²⁾. 먼저 1KByte 메모리를 가지는 MiFare Classic 형식(MiFare Classic 1K)의 태그를 NFC 쉘드를 통해 읽는 예를 살펴보자. MiFare Classic 1K 태그의 메모리는 0번에서 15번까지 16개의 섹터(sector)로 이루어지며 각 섹터는 0번부터 3번까지 4개의 블록(block)으로 이루어지므로 각 블록은 16바이트 크기를 가지게 된다. (4 블록 × 16 섹터 × 16 바이트 = 1024 바이트) 이 중 0번 섹터의 0번 블록은 제조사 블록 또는 제조사 데이터라고 불리며 여기에 태그를 유일하게 구별할 수 있는 UID(Unique Identifier)가 기록되어 있다.³⁾ 스케치 1은 MiFare Classic 태그의 UID를 읽어서 시리얼 모니터로 출력하는 예이다.

스케치 1. MiFare Classic 태그 UID 읽기

```
#include <Wire.h>
#include <Adafruit_PN532.h>

#define PN532_IRQ      2           // Interrupt Request
#define PN532_RESET    3           // 디폴트로 연결되어 있지 않음

// NFC 쉘드 객체 생성
Adafruit_PN532 nfc(PN532_IRQ, PN532_RESET);

void setup(void) {
  Serial.begin(115200);
  Serial.println("Hello!");
```

1) 이미지 출처: <http://www.adafruit.com/product/789>

2) <https://github.com/adafruit/Adafruit-PN532/>

3) 보다 자세한 내용은 ISO/IEC 14443-3 기술문서를 참고하도록 한다.

```
nfc.begin();                                // NFC 쉘드 초기화

uint32_t versiondata = nfc.getFirmwareVersion(); // 버전 정보 확인
if (!versiondata){
    Serial.print("Didn't find PN53x board");
    while(1);
}

Serial.print("Found chip PN5");
Serial.println((versiondata >> 24) & 0xFF, HEX);        // 칩 번호
Serial.print("Firmware ver.");
Serial.print((versiondata >> 16) & 0xFF, DEC);          // Major 버전
Serial.print('.');
Serial.println((versiondata >> 8) & 0xFF, DEC);         // Minor 버전

nfc.SAMConfig();                                    // NFC 쉘드를 읽기 상태로 설정

Serial.println("Waiting for an ISO14443A Card ...");
}

void loop(void) {
    uint8_t success;
    uint8_t uid[] = { 0, 0, 0, 0, 0, 0, 0, 0 };    // UID 저장 버퍼
    // UDI 길이 (MiFare Classic은 4 바이트, MiFare Ultralight는 7 바이트)
    uint8_t uidLength;

    // 태그 검사
    success = nfc.readPassiveTargetID(PN532_MIFARE_ISO14443A, uid, &uidLength);

    if (success){
        Serial.println("Found an ISO14443A card");
        Serial.print("  UID Length : ");
        Serial.print(uidLength, DEC);
        Serial.println(" bytes");
        Serial.print("  UID Value : ");
```

```
nfc.PrintHex(uid, uidLength);      // UID 출력  
  
}  
delay(1000);  
}
```

그림 8은 스케치 1의 실행 결과로 NFC 칩의 안테나 부분에 태그를 가져가면 해당 UID가 출력되는 것을 확인할 수 있다. 그림 8에서 첫 번째 UID는 카드 타입의 태그에 해당하며, 두 번째 UID는 스마트폰을 카드 모드로 설정한 후 NFC 칩에 접촉하였을 때의 결과이다.



그림 8. 스케치 1 실행 결과

MiFare Classic 1K 태그에서 섹터 0번의 구성은 그림 9와 같다. 다른 섹터 역시 0번 블록이 데이터 기록을 위해 사용된다는 점을 제외하면 동일하다.

섹터	블록	블록 내 바이트 번호															설명	
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14		15
0	0																	생산자 블록
	1																	데이터
	2																	데이터
	3	Key A					Access Bit GPB			Key B					섹터 트레일러			

그림 9. MiFare Classic 1K 태그의 섹터 구성

각 섹터의 3번 블록은 섹터 트레일러 (sector trailer) 블록이라고 불리며 섹터의 정보 접근에 필요한 정보가 기록된다. 접근 비트(access bit)의 설정에 따라 판독기는 Key A나 Key B를 통해 인증을 수행하여야 한다. 9번 바이트는 범용 바이트(General Purpose Byte)라고 불리며 **MiFare Classic 태그에서는 사용되지 않는다**. 접근 비트는 3바이트 24비트로 이 중 12비트는 나머지 12비트의 반전된 정보가 기록되어 있으므로 실제 정보는 12비트만 포함되어 있다. MiFare Classic 태그의 경우 모든 키 값은 디폴트로 0xFFFFFFFF이 설정되어 있다. 이 장에서 사용하는 태그는 MiFare Classic 1K 태그이며 태그의 종류 및 메모리 크기에 따라 블록을 읽는 방법은 달라질 수 있으므로 해당 문서를 참조하여야 한다.

스케치 2. MiFare Classic 태그 내용 읽기

```
#include <Wire.h>
#include <Adafruit_PN532.h>
#define USE_KEY_B    1           // 인증에 사용할 키
#define USE_KEY_A    0

#define PN532_IRQ     2           // Interrupt Request
#define PN532_RESET   3           // 디폴트로 연결되어 있지 않음

// NFC 실드 객체 생성
Adafruit_PN532 nfc(PN532_IRQ, PN532_RESET);

void setup(void) {
  Serial.begin(115200);
```

```
Serial.println("Hello!");

nfc.begin();                                // NFC 쉘드 초기화

uint32_t versiondata = nfc.getFirmwareVersion(); // 버전 정보 확인
if (!versiondata){
    Serial.print("Didn't find PN53x board");
    while(1);
}

Serial.print("Found chip PN5");
Serial.println((versiondata >> 24) & 0xFF, HEX);    // 칩 번호
Serial.print("Firmware ver.");
Serial.print((versiondata >> 16) & 0xFF, DEC);       // Major 버전
Serial.print('.');
Serial.println((versiondata >> 8) & 0xFF, DEC);      // Minor 버전

nfc.SAMConfig();                              // NFC 쉘드를 읽기 상태로 설정

Serial.println("Waiting for an ISO14443A Card ...");
}

void loop(void) {
    uint8_t success;
    uint8_t uid[] = { 0, 0, 0, 0, 0, 0, 0, 0 }; // UID 저장 버퍼
    // UDI 길이 (MiFare Classic은 4 바이트, MiFare Ultralight는 7 바이트)
    uint8_t uidLength;
    uint8_t currentblock;                      // 현재 블록 번호
    bool authenticated = false;                // 인증 여부
    uint8_t data[16];                          // 블록 데이터 저장 버퍼

    // NDEF와 MiFare Classic에서 Key B는 0xFFFFFFFF로 동일함
    uint8_t key[6] = { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF };
```

```
    success = nfc.readPassiveTargetID(PN532_MIFARE_ISO14443A, uid,
    &uidLength);

    if(success){
        Serial.println("Found an ISO14443A card");
        Serial.print("  UID Length : ");
        Serial.print(uidLength, DEC);
        Serial.println(" bytes");
        Serial.print("  UID Value : ");
        nfc.PrintHex(uid, uidLength);           // UID 출력

        // 4개 블록을 포함하는 16개 섹터, 총 64개 블록 데이터 읽기
        // 섹터별로 인증을 수행
        for (currentblock = 0; currentblock < 64; currentblock++){
            bool isFirstBlock = nfc.mifareclassic_IsFirstBlock(currentblock);

            if(isFirstBlock){
                Serial.print("-----Sector ");
                Serial.print(currentblock / 4, DEC);
                Serial.println("-----");

                authenticated = false;
            }

            // 섹터의 첫 번째 블록에서 인증
            if (!authenticated){
                // MiFare Classic의 Key A 디폴트값 : FF FF FF FF FF FF
                // NDEF의 Key A 디폴트값 : A0 A1 A2 A3 A4 A5
                // MiFare Classic & NDEF의 Key B 디폴트값 : FF FF FF FF FF FF
                success = nfc.mifareclassic_AuthenticateBlock (uid, uidLength,
                currentblock, USE_KEY_B, key);

                if (success) authenticated = true;
            }
        }
    }
```

```
Serial.print("Block ");
Serial.print(currentblock, DEC);
if(currentblock < 10) Serial.print(" ");
else Serial.print(" ");

if (!authenticated){
    Serial.println(" unable to authenticate");
}
else{
    // 블록 데이터 읽기
    success = nfc.mifareclassic_ReadDataBlock(currentblock, data);

    if(success){
        nfc.PrintHexChar(data, 16);    // 16 바이트 데이터 출력
    }
    else{
        Serial.println(" unable to read");
    }
}
}
else{
    Serial.println("This doesn't seem to be a Mifare Classic card!");
}

// 태그 스캔을 위해 문자 입력 대기
Serial.println("\n\nSend a character to run the mem dumper again!\n\n");

while (!Serial.available());
while (Serial.available()) Serial.read();
}
```

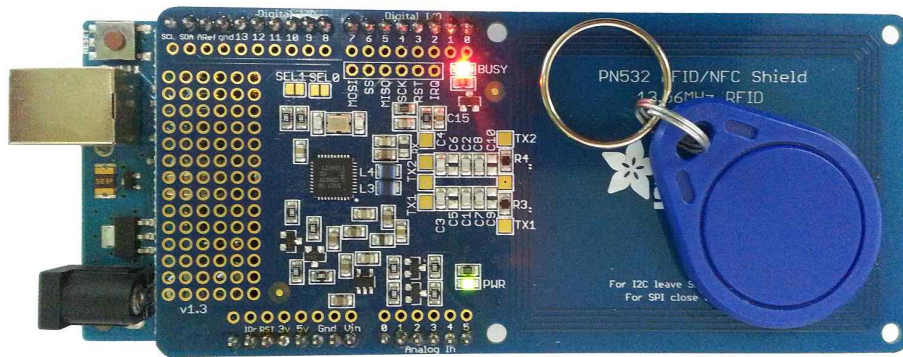


그림 10. NFC 설드로 태그 내용 읽기

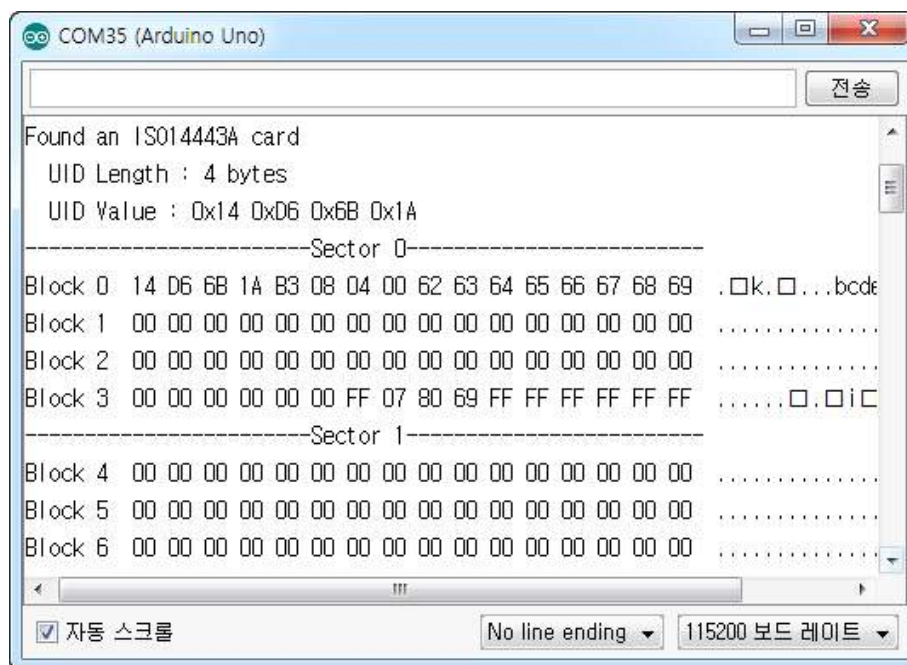


그림 11. 스케치 2 실행 결과

마지막으로 MiFare Classic 1K 태그에 데이터를 기록하는 방법을 살펴보자. 내용을 기록하기 위해서도 인증 과정이 필요하며 `mifareclassic_WriteDataBlock` 함수를 통해 블록에 데이터를 기록할 수 있다. 스케치 3은 4번 블록에 데이터를 기록하고 이를 다시 읽어서 출력하는 스케치의 예이다.

스케치 3. MiFare Classic 태그 내용 쓰기

```
#include <Wire.h>
#include <Adafruit_PN532.h>
#define USE_KEY_B    1           // 인증에 사용할 키
#define USE_KEY_A    0

#define PN532_IRQ     2           // Interrupt Request
#define PN532_RESET   3           // 디폴트로 연결되어 있지 않음

// NFC 쉘드 객체 생성
Adafruit_PN532 nfc(PN532_IRQ, PN532_RESET);

void setup(void) {
  Serial.begin(115200);
  Serial.println("Hello!");

  nfc.begin();                      // NFC 쉘드 초기화

  uint32_t versiondata = nfc.getFirmwareVersion(); // 버전 정보 확인
  if (!versiondata){
    Serial.print("Didn't find PN53x board");
    while(1);
  }

  Serial.print("Found chip PN5");
  Serial.println((versiondata >> 24) & 0xFF, HEX);      // 칩 번호
  Serial.print("Firmware ver.");
  Serial.print((versiondata >> 16) & 0xFF, DEC);        // Major 버전
  Serial.print('.');
  Serial.println((versiondata >> 8) & 0xFF, DEC);        // Minor 버전

  nfc.SAMConfig();                      // NFC 쉘드를 읽기 상태로 설정

  Serial.println("Waiting for an ISO14443A Card ...");
}
```

```
void loop(void) {
    uint8_t success;
    uint8_t uid[] = { 0, 0, 0, 0, 0, 0, 0, 0 }; // UID 저장 버퍼
    // UDI 길이 (MiFare Classic은 4 바이트, MiFare Ultralight는 7 바이트)
    uint8_t uidLength;
    uint8_t data[16] = ""; // 블록 데이터 저장 버퍼
    char testString[] = "Write Test"; // 블록에 기록할 데이터
    int block_no = 4; // 데이터를 기록할 블록 번호

    // MiFare Classic에서 Key A와 Key B는 0xFFFFFFFF로 동일함
    uint8_t key[6] = { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF };

    success = nfc.readPassiveTargetID(PN532_MIFARE_ISO14443A, uid,
    &uidLength);

    if(success){
        Serial.println("Found an ISO14443A card");
        Serial.print("  UID Length : ");
        Serial.print(uidLength, DEC);
        Serial.println(" bytes");
        Serial.print("  UID Value : ");
        nfc.PrintHex(uid, uidLength); // UID 출력

        success = nfc.mifareclassic_AuthenticateBlock(uid, uidLength, 4,
        USE_KEY_A, key);

        if(success){
            memcpy(data, testString, sizeof(testString));
            Serial.print("\n** Write block data \");
            Serial.print(testString);
            Serial.println(String("\n" to block ") + block_no);

            // 블록 데이터 쓰기
```

```
success = nfc.mifareclassic_WriteDataBlock (block_no, data);

if(success){
    // 블록 데이터 읽기
    success = nfc.mifareclassic_ReadDataBlock(block_no, data);

    if(success){
        Serial.println(String("** Read block data from block ") + block_no);
        nfc.PrintHexChar(data, 16);
    }
    else Serial.println("** Read Error");
}
else{
    Serial.println("** Write Error");
}
}
else{
    Serial.println("** Authentication Error");
}
}

// 태그 쓰기/읽기를 위해 문자 입력 대기
Serial.println("\nSend a character to run write/read again!\n\n");

while (!Serial.available());
while (Serial.available()) Serial.read();
}
```

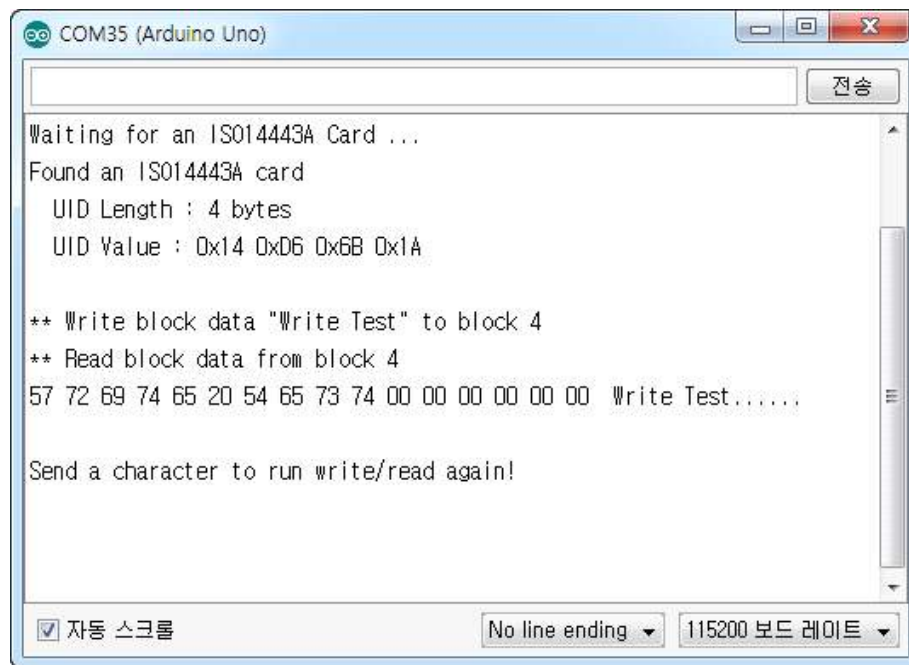


그림 12. 스케치 3 실행 결과