안드로이드 씽스 입문자 키트

메카솔루션

Contents

1 소개

사물 인터넷과 4차 산업 혁명 시대를 맞으면서, 이와 관련된 다양한 플렛폼과 솔루션들이 등장하고 발전하고 있습니다. 과거 휴대용 전화기에 불과했던 휴대폰이 이제는 멀티코어 프로세서와 그래픽 칩까지 달린 어엿한 컴퓨터인 스마트 폰으로 바뀌었고, 손목 시계도 마친가지로 스마트 워치로, TV도 지상파/케이블을 시간에 맞춰 보던 것이 이제는 내가 보고 싶을 때 보고 싶은 영상을 볼 수 있는 스마트 TV로 바뀌었습니다. 콘솔 게임기(Xbox, Play Staytion 등)와 아케이드(오락실) 게임기도 인터넷에 연결되어 다른 지역에 있는 사람과 함께 할 수 있게 되었습니다. 스마트 홈 오피스라 하여 전등이나 스위치, 콘센트에도 인터넷이 장착되고 있습니다. 강아지나 아이들이 있는 집에는 인터넷 카메라를 설치합니다. 앞으로 청소기, 세탁기 같은 가전부터 자동차나 비행기, 배, 공장의 거대한 기계들까지 점점 더 많은 사물들이 인터넷에 연결되게 될 것입니다.

오늘날에는 4G/LTE를 이용하면 전국 어디서나 빠른 인터넷을 이용할 수 있습니다. 또한 WiFi를 제공하는 시설도 흔하게 만날 수 있습니다. 구글에서는 수 조의 비용을 들여서 전세계를 인공위성과 열기구로 뒤덮어 모든 사람들에게 WiFi를 제공하겠다는 룬(Loon) 프로젝트를 발표했습니다. 이제는 인터넷 없이 살 수 없는 세상이 되었고, 앞으로는 인터넷에 연결되지 않은 사물이 없는 세상이 올 것입니다.

구글이 4차 산업 혁명 시대에 따라 제공하는 플렛폼 중 하나가 안드로이드 씽스(Android Things)입니다. 안드로이드 씽스는 싱글 보드 컴퓨터로 유명한 라즈베리파이나 인텔의 에디슨 등에 안드로이드 씽스 운영체제를 설치하면 안드로이드 앱을 이용할 수 있게 해주는 플렛폼입니다. 다만 기존의 스마트폰에 사용되는 안드로이드 운영체제와 다른 점은, 안드로이드 씽스는 사물 인터넷 솔루션이기 때문에 라즈베리파이 등을 이용해 만들 사물을 사용하기 위한 하나의 앱만 가진다는 점입니다. 가장 큰 변화는 기존의 앱 개발자들이 앱을 개발하듯이 이제는 장치를 개발할 수 있다는 점입니다. 구글에서 안드로이드 씽스를 설명할 때 사용하는 다음과 같은 문구를 사용합니다.

If you can build an app, you can build a device

"당신이 앱을 만들 수 있으면, 장치도 만들 수 있습니다."

안드로이드 씽스를 통해 스타트업 기업들도 안드로이드 앱 수준의 안정성과 품질을 가진 제품을 만들 수 있습니다. 라즈베리파이와 함께 4차 산업 혁명을 이끌어 가고 있는 아두이노의 경우 쉽고 빠르게 배울 수 있다는 장점이 있지만, 성능이나 기능 면에서 8-bit 마이크로 컨트롤러의 한

계를 넘을 수가 없습니다. 물론 단순한 사물 인터넷 장치를 만드는데에는 아두이노가 더 적합합니다. 하지만, 영상 및 소리의 입출력이나 처리, 각종 USB장치의 이용 등에 있어서는 크기나 비용면에서 라즈베리파이를 이길 수 없습니다. 특히 알파고의 등장으로 유명해진 딥러닝도 아두이노에서는 제한되지만, 라즈베리파이에서는 기존 안드로이드 앱의 적용이 가능하다는 부분 때문에 사실상 입증이 되었습니다. 안드로이드 씽스 GitHub에는 영상처리 예제도 이미 올라와 있습니다. 딥러닝은 또한 사물 인터넷과 밀접한 관계가 있습니다. 우선 사물 인터넷은 사물이 인터넷에 연결되어 있기 때문에 사용자만 동의한다면, 지속적으로 데이터를 수집하고 축적할 수 있습니다. 딥러닝은 말 그대로 학습이기 때문에 많은 데이터를 필요로 하게 됩니다. 사물 인터넷이 보다 활성화 된다면, 이를 통한 데이터 수집으로 보다 다양하고 똑똑한 딥러닝이 가능하게 됩니다.

단점이라면, 교육용인 아두이노와 달리 안드로이드 개발은 상용화 소프트웨어 개발이기 때문에, 코드가 복잡하고 어려운 편입니다.

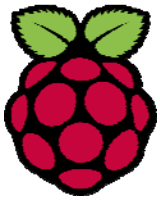
과거 그리고 현재에도 물건을 판매하거나 구매하면 거기서 끝이 되게 됩니다. 판매자로써는 구매자가 만족을 하였는지 불편한 점은 없는지, 또한 구매자는 물건의 효율이나 문제해결에 있어서 서로 연락을 하지 않으면 알기가 어렵습니다. 사물 인터넷이 보다 활성화된다면, 판매자와 구매자가 별다른 노력을 하지 않아도 사물 스스로가 판매자 또는 생산자와 지속적으로 정보를 주고 받으며 개선 방법을 찾기 훨씬 쉬워질 것입니다.

이 책을 통해 여러분들이 이런 시대를 준비할 수 있도록 사물 인터넷 솔루션인 안드로이드 씽스를 활용해 간단한 버튼이나 LED를 다는 것부터 온도 센서 등의 입력을 받거나 모터를 돌리는 것까지 기초과정을 통해 단순한 장치를 만들 수 있는 소양을 갖출 수 있도록 하고 합니다. 또한 앞서 말씀드린 것처럼 안드로이드 앱 개발까지 함께 배울 수 있습니다.

2 안드로이드 씽스 이해하기

본격적으로 시작하기 앞서 라즈베리파이와 안드로이드에 대해 정리해보겠습니다.

2.1 라즈베리파이란?



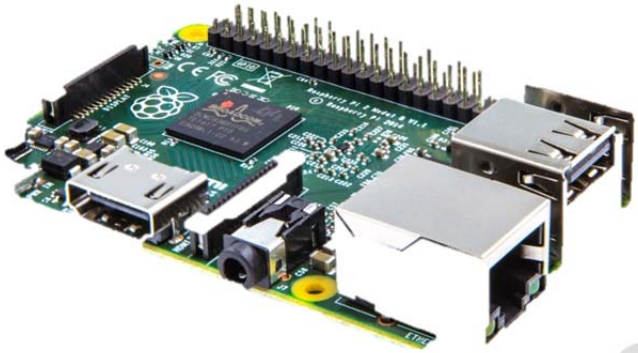
라즈베리파이(Raspberry Pi)는 영국의 라즈베리파이 재단에서 후진국/개도국 학생들이 쉽게 코딩을 접하고 사용할 수 있도록 교육용으로 만든 초소형/초저가 PC입니다. 최신형인 3모델 B를 기준으로 하드웨어는 ARM 기반의 Broadcom에서 만든 CPU/GPU를 중심으로 이더넷과 WiFi가 내장되어 있고, 블루투스와 HDMI, USB 포트 등이 있습니다. 자세한 사양은 아래와 같습니다.

사양

SoC	Broadcom BCM2837
CPU	1.2GHz ARM Cortex-A53 MP4
GPU	Broadcom VideoCore IV MP2 400 MHz
메모리	1GB LPDDR2
네트워크	10/100 Mbps 이더넷 WiFi 내장 802.11n 블루투스 4.1
영상출력	컴포지트/HDMI/DSI
음성출력	3.5mm 잭, HDMI, I2S
USB	USB 2.0 x 4포트
GPIO	40핀
규격	85.60 x 56.5mm, 45g
SD카드 슬롯	Micro SD
제조사	RS Component(Sony), UK/PremierFarnell(Element14-Embest), UK

이 책에서는 다루지 않으나 일반적으로 라즈베리파이는 모니터/키보드/마우스를 연결한 후 라즈비안이라는 리눅스 기반의 전용 운영체제를 설치해서 하나의 독립적인 컴퓨터로 사용합니다. 리눅스와 파이썬, 그리고 리눅스를 통한 인터넷과 네트워크, 웹서버 개발, 관리 등을 배우는데 매우 용이합니다. 입출력을 제거한 후 SSH나 원격접속을 이용하면 소형 및 이동형 시스템에도 사용할 수 있습니다. 특히 사물 인터넷 시대를 맞이하여 각종 인터넷 통신 방법과 웹프로그래밍 능력을 갖추면, 훨씬 깊이 있는 시스템을 개발할 수 있기 때문에, 라즈베리파이를 통해 리눅스를 배우

는 것도 매우 추천합니다.



무엇보다 아두이노와 함께 라즈베리파이의 장점은 그 인기로 인해, 인터넷에 자료가 풍부하다는 점입니다. 라즈베리파이 자체 뿐 아니라, 리눅스도 필요한 자료를 얻기가 매우 쉬워서 아주 특이한 프로젝트를 하지 않는 이상 대부분 여러분들이 구현하고자 하는 기능에 대해 시도하고, 질물하며, 완성한 사람이 있는 경우가 매우 많습니다.

2.2 안드로이드 씽스란?

안드로이드 씽스(Android Things)는 안드로이드(Android)와 사물 인터넷(Internet of Things)를 합친 말로, 구글에서 기존의 안드로이드 운영체제가 스마트 폰, 타블렛, 웨어러블, TV, 오토까지 이용되던 것을 넘어서, 장치 개발에도 사용할 수 있도록, 단순화시킨 운영체제입니다. 장치 개발이기 때문에, 기존 안드로이드 운영체제와 달리 스크린 없이 사용할 수 있으며, 무엇보다 가장 큰 차이는 장치를 사용하기 위한 앱 하나만 구동한다는 점입니다. 그러한 동시에 기존에 안드로이드 앱에서 이용할 수 있었던 구글 플레이 서비스, 파이어 베이스, 클라우드 등 다양한 구글 서비스에서 제공하는 모든 API도 사용할 수 있습니다.

사실 안드로이드 씽스가 아니어도 기존에 라즈베리파이나 기타 ARM 보드에 안드로이드 운영체제를 설치해서 사용하는 방법은 있었지만, 구글에서 공식적으로 지원하는 것이 아니었기 때문에, 위에서 말하는 구글에서 제공하는 서비스는 상당히 제한이 되었습니다. 무엇보다 안드로이드

씽스는 부팅 후 원하는 앱을 찾아가서 실행할 필요 없이, 곧바로 앱이 실행되게 되므로 장치 개발에 더욱 적합하다고 할 수 있습니다.

안드로이드 기반이기 때문에, 장치의 의존성 즉 앱이 필요로 하는 소프트웨어가 안드로이드와 장치에 모두 연결되어 있다면, 하나의 앱을 안드로이드와 안드로이드 씽스 모두에서 사용할 수도 있습니다. 가령 라즈베리파이에 카메라와 터치스크린을 연결한다면 기존의 안드로이드 앱을 안드로이드 씽스에서도 사용할 수 있고 그 반대도 가능합니다. 또한 안드로이드 씽스 외에도 인텔의 에디슨, 쥴이나 NXP 모듈을 사용할 수도 있습니다. 라즈베리파이3는 쉽게 저렴하게 구할 수 있다는 장점이 있지만, 소형화나 대량 생산에는 적합하지 않기 때문에 구글에서도 소형/대량 제품 개발에는 에디슨이나 NXP 모듈을 사용할 것을 권하고 있습니다. 하지만 에디슨은 가격이 비싼 편이고, NXP는 아직 국내에서 구하기 쉽지 않기 때문에 개인 사용자 기준으로는 역시 라즈베리파이가 낫습니다. 추가로 안드로이드 씽스 자체가 아직 개발 버전(2017년 09월 21일 현재 Developer Preview 5.1)이기 때문에, 미완성된 부분도 있지만, 나중에 정식 버전이 나오면 안드로이드 씽스를 지원하는 보드도 더 많고 다양해질 것입니다.

2.3 안드로이드 앱 개발

안드로이드 앱 개발은 원래 ADT(Android Development Tool, 안드로이드 개발 도구)를 기반으로 대표적인 IDE(;Integrated Development Environment, 통합 개발 환경)인 이클립스를 통해 개발이 이루어져왔고, 지금도 그렇게 진행하는 개발자들이 많습니다. 하지만 2015년도 6월부터 구글에서 이클립스에 대한 지원을 중단하고 자체 IDE인 안드로이드 스튜디오에 집중하겠다고 발표한 이후로, 새로 시작하거나 이클립스에서 옮겨온 개발자들은 안드로이드 스튜디오를 통해 안드로이드 앱을 개발하고 있습니다.

과거 이클립스를 활용할 때는 ADT와 함께 다른 개발 도구와 연동해서 개발을 했기 때문에 사용하는 도구에 따라 개발 방법이 다소 다르고 서로 정보를 나누기 어려운 부분이 있었으나, 안드로이드 스튜디오로 통합되면서 한 가지 방법만 알면 되므로 배우고 개발하기가 더욱 쉬워졌습

니다. 그러나 비교적 쉬워진 것일 뿐이고, 안드로이드 앱 개발 자체가 코딩 교육보다는 구글 플레이 등을 통해 배포되는 상용 소프트웨어 개발이 목적이기 때문에, 아두이노와 비교하면 굉장히 단순한 기능을 구현하는 것도 코드가 어렵고 복잡한 편입니다. 특히 하나의 파일만 다뤄도 충분한 아두이노와 달리 안드로이드는 기본적으로 메인 엑티비티부터 그래들과 매니페스트까지 2~3개 이상의 파일을 만져줘야 하고, 프로젝트의 규모와 기능에 따라 다뤄야 하는 파일이 매우 많아질 수 있습니다. 코드도 기본적으로 자바와 객체 지향에 대한 지식이 어느 정도는 필요하기 때문에 초보자가 다루기는 제한이 있습니다. 이런 문제에 대해서 애플의 경우 아이폰 앱 개발에 원래 오브젝티브 C를 사용하던 것에서 자체 언어인 스위프트(SWIFT)로 옮겨가면서 개발이 많이 용이해졌는데, 마찬가지로 구글에서도 자바에서 코틀린(Kotlin)이란 자체 언어로 옮겨가려고 시도중입니다.

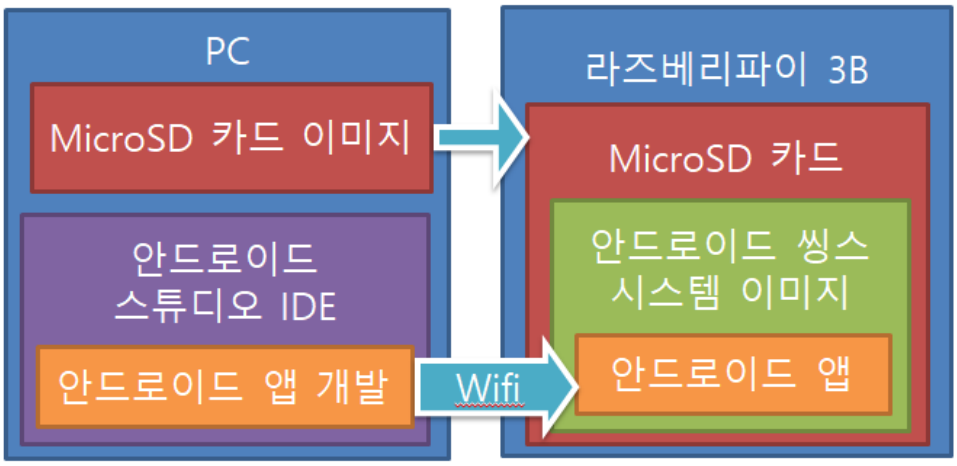
이 책에서는 이런 어려운 부분에 대해서 최대한 쉽게 넘기고, 라즈베리파이와 안드로이드의 기능을 활용해서 예제를 완성해보는 것에 최대한 집중하여, 독자들이 안드로이드 씽스가 어떤 것인지 경험시켜 드리도록 하고자 합니다.

3 안드로이드 씽스 준비하기

안드로이드 씽스를 시작하기 위해서는 다음과 같은 준비물이 필요합니다.

라즈베리파이 3B	
MicroUSB 전원	2A 이상
MicroSD 카드	
MicroSD 카드 리더기	
모니터	필수는 아닙니다.
FTDI 시리얼 컨버터	시리얼 연결시
공유기 + 랜선	공유기를 통해 연결시
랜카드(이더넷 포트) + 랜선	랜선으로 직접 연결시

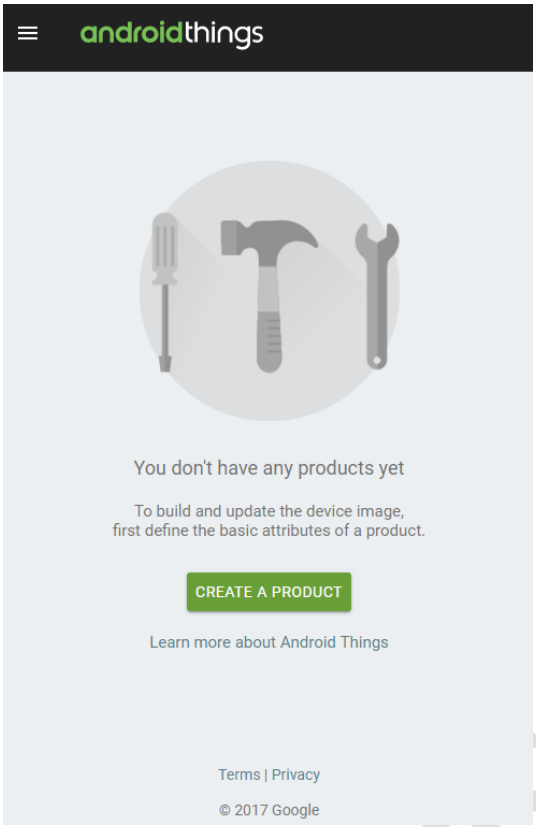
안드로이드 씽스의 구조와 개발은 다음과 같은 형태로 이루어지게 됩니다. 우선 안드로이드 씽스 시스템 이미지를 PC에서 SD카드에 입력한 후 SD카드를 라즈베리파이에 연결해줍니다. 안드로이드 씽스 시스템 이미지가 준비된 라즈베리파이를 부팅시킨 후, 시리얼 통신 등을 이용해 WiFi에 연결해줍니다. 다시 PC로 돌아와서 안드로이드 스튜디오에서 원하는 안드로이드 앱을 개발합니다. 이를 WiFi 연결을 통해 안드로이드 씽스에 설치해주면, 안드로이드 앱이 작동하게 됩니다.



또한 다음과 같은 소프트웨어들이 필요합니다.

안드로이드 스튜디오	2.3 이상
SD card Formatter	
Win32DiskImager	
Putty	시리얼 연결시
DHCP server for windows	랜선으로 직접 연결시

3.1 안드로이드 씽스 시스템 이미지 준비하기

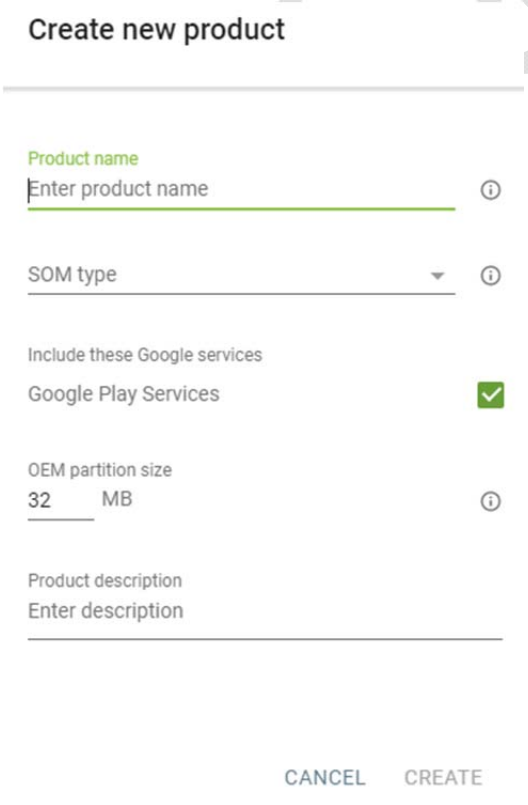


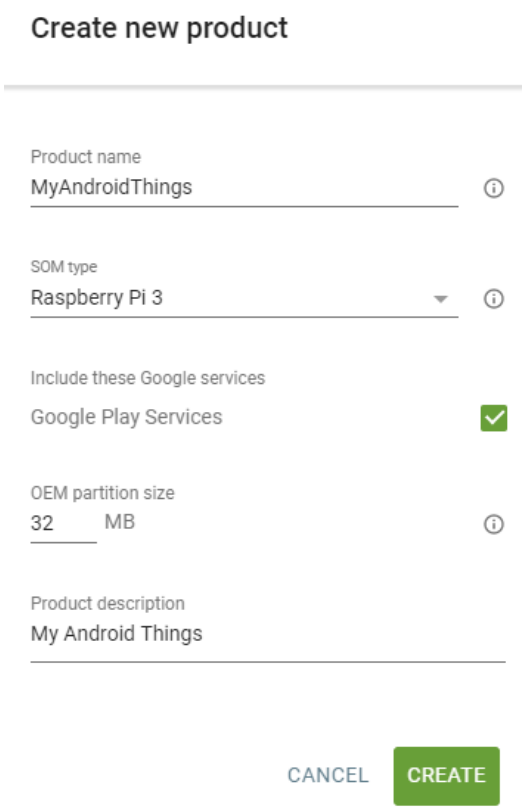
먼저 안드로이드 씽스 콘솔 홈페이지로 갑니다. 안드로이드 씽스 콘솔을 이용하기 위해서는 구글 계정이 필요합니다.

phttps://partner.android.com/things/console/

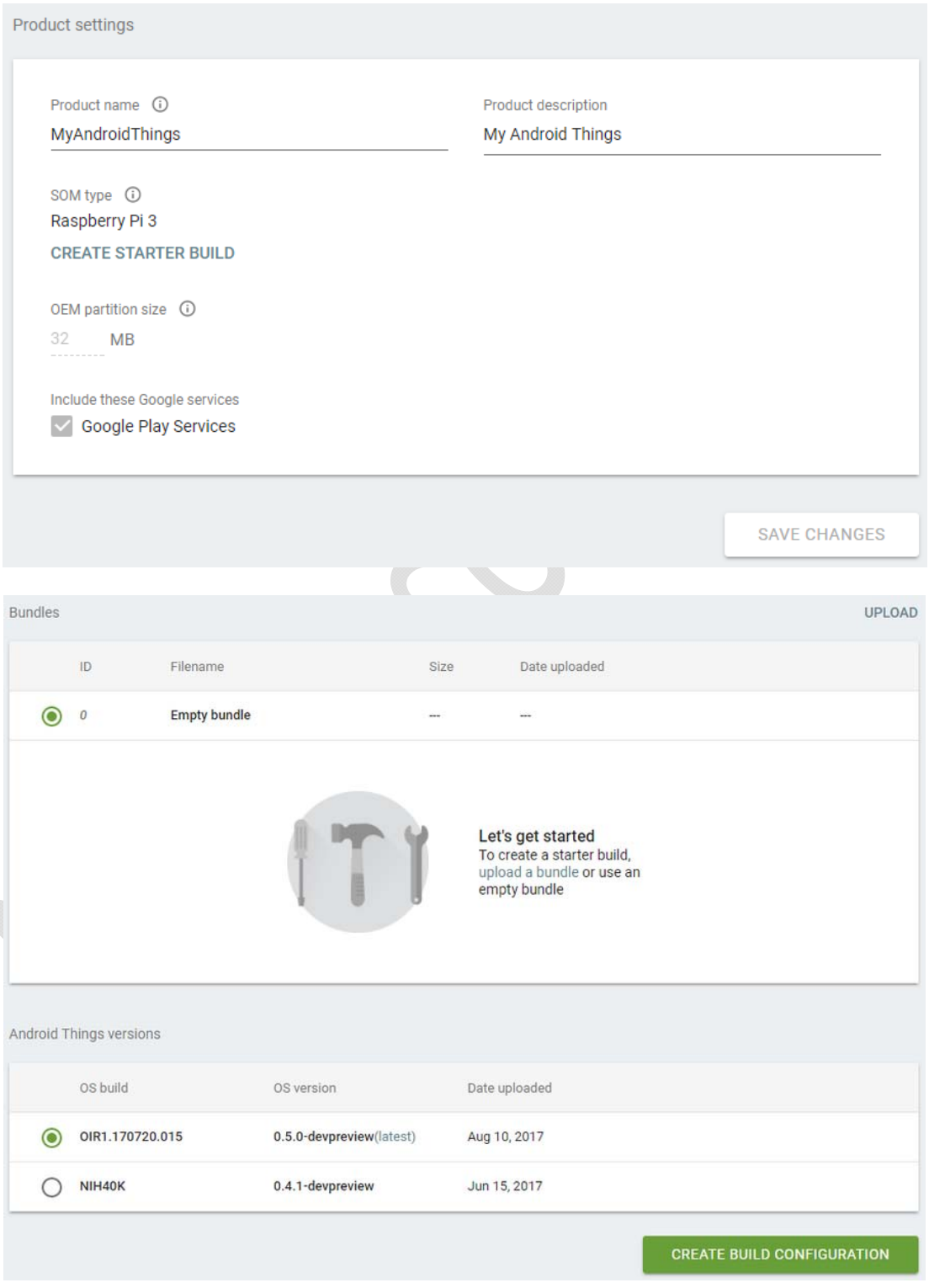
시작 과정은 안드로이드 씽스를 시작하는 과정은 라즈비안과 비슷합니다. 다만 안드로이드 씽스 콘솔 사이트에서 자신의 보드(여기서는 라즈베리파이)를 선택해서 이에 맞는 시스템 이미지를 생성하여 다운 받아야 한다는 점입니다. 언뜻 보면 라즈비안에 비해 불편해 보이지만, 앞서 말씀드린대로 안드로이드 씽스는 사용 소프트웨어 개발을 목적으로 하기 때문에 이와 관련하여 시스템 이미지와 앱 업데이트 및 배포를 위한 목적으로 콘솔 시스템을 이용해야 합니다.

CREATE A PRODUCT 를 클릭합니다.

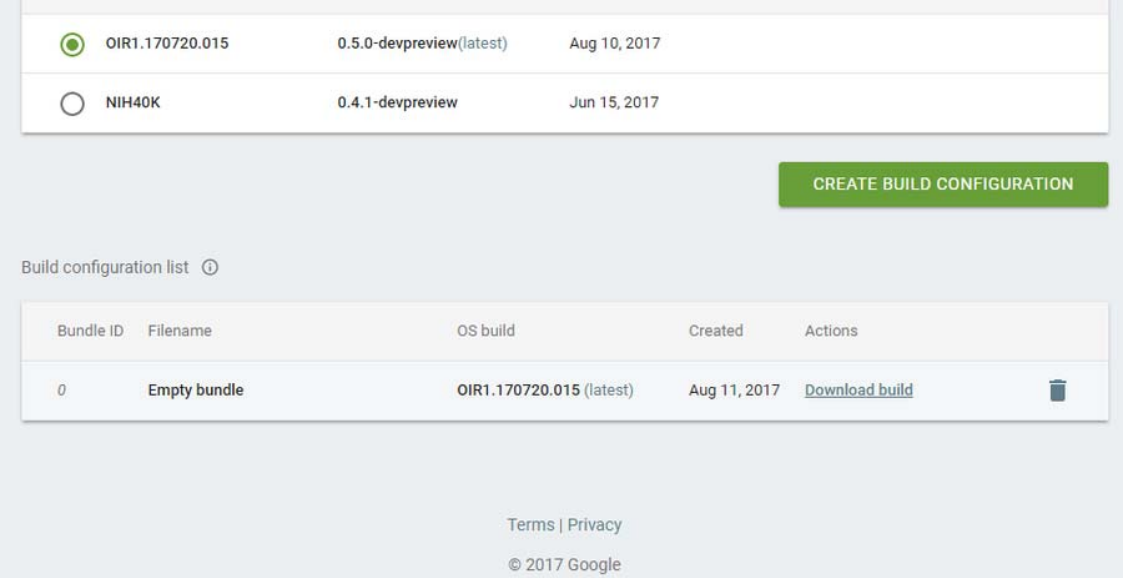




빈 칸을 위와 같은 식으로 채워줍니다. SOM type만 Raspberry Pi 3로 하는 것을 제외하면 원하는데로 하셔도 됩니다. 완료되면 CREATE을 누릅니다. 프로덕트가 생성되면 위와 같은 화면이 뜹니다. CREATE STARTER BUILD를 눌러줍니다.



위 화면이 뜨면 오른쪽 아래 CREATE BUILD CONFIGURATION을 클릭합니다.



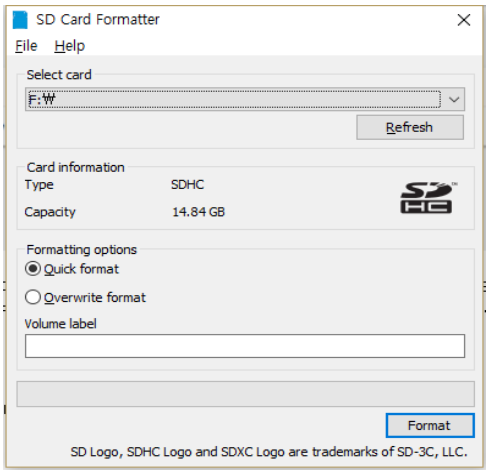
바로 아래에 Empty bundle이 생성됩니다. 오른쪽에 Download build를 누르면 build를 시작해서 3~5분 정도 후에 다운로드가 자동으로 시작됩니다.다운로드가 완료되면 다운로드 받은 폴더로 가서 압축을 풉니다. 하지만 시간이 다소 걸리므로 빌드를 하고 다운로드를 하는 동안, 이미지를 굽는데 필요한 유틸들을 준비하겠습니다.

3.2 SD 카드에 시스템 이미지 굽기

먼저 SD카드를 포맷하는데에는 SDFormatter가, 시스템 이미지를 굽기 위해서는 Win 32 Disk Imager가 필요합니다. 각각을 다음 사이트에서 다운 받습니다.

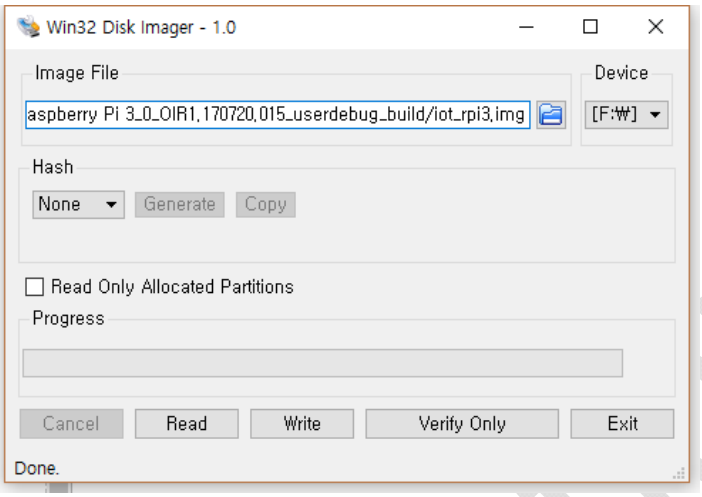
https://www.sdcard.org/downloads/formatter_4/

https://sourceforge.net/projects/win32diskimager/

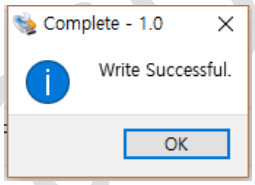


프로그램을 설치하고 SD 카드를 컴퓨터에 연결한 다음 프로그램을 실행시킵니다. 새 SD카드가 아닌 기존에 라즈베리파이용으로 라즈비안이나 안드로이드 씽스를 사용하던 SD카드의 경우 Linux용과 부트용으로 파티션이 2개로 나눠져 있습니다. Linux용은 윈도우에서 접근할 수 없기 때문에 빈 드라이브로 뜨고, 부트용은 라즈비안의 boot, 안드로이드 씽스는 rpiboot로 드라이브 이름이 뜹니다. 이때 빈 드라이브를 포맷해야 합니다. 빈 드라이브

를 포맷하면 부트 드라이브는 자동으로 없어지게 됩니다. 연결한 SD 카드에 해당하는 드라이브를 선택한 후 Format을 누릅니다. 마찬가지로 Win 32 Disk Imager를 설치 실행합니다.



먼저 안드로이드 씽스 콘솔에서 다운로드 받은 압축 파일을 풀면 iot_rpi3.img가 있습니다. 이 파일을 위와 같이 Win 32 Disk Imager에서 선택하고, 앞에서 포맷한 SD카드에 안드로이드 씽스 이미지를 구워줍니다. 이미지 굽기가 완료되면 아래와 같은 창이 뜹니다.

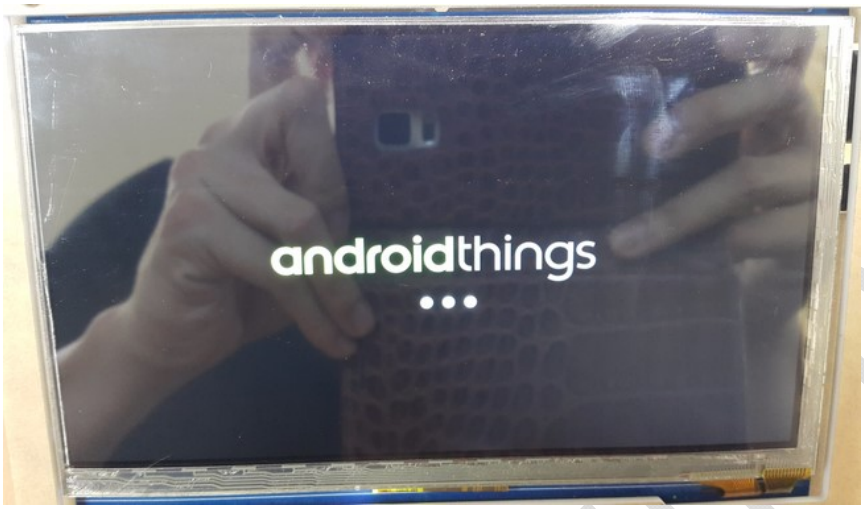


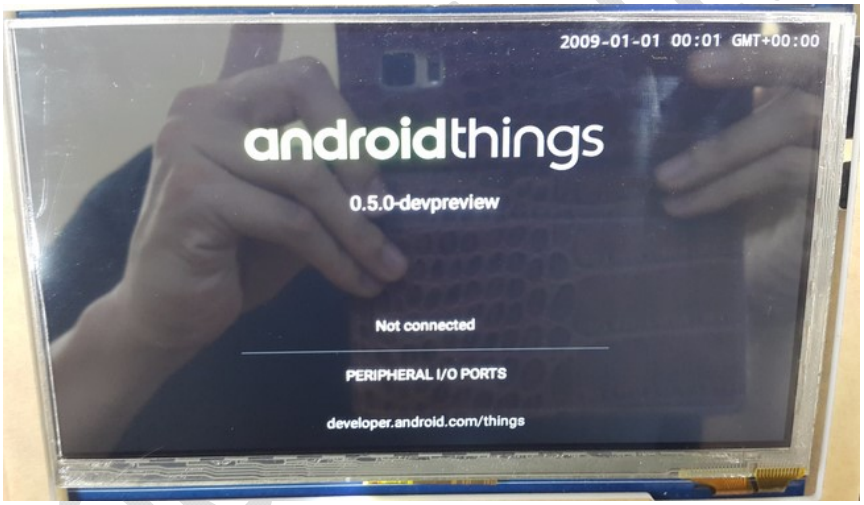
완료되면 SD카드를 안전하게 제거한 후, 라즈베리파이에 꽂아줍니다. 라즈베리파이의 SD카드 슬롯은 SD카드가 장착된 후에 약간 돌출되게 되어있는데, 보드를 다루거나, 케이스를 탈부착하는 등의 과정에서 SD카드가 부러지는 일이 간혹 발생하기 때문에 SD카드가 장착된 라즈베리파이를 다룰 때 이점을 유의하시고, 각별히 조심하시기 바랍니다.

3.3 안드로이드 씽스 부팅

안드로이드 씽스는 하나의 안드로이드 앱이 라즈베리파이나 기타 안드로이드 씽스 지원 하드웨어에서 작동할 수 있도록 지원해주는 역할에 불과하고, 일반적인 운영체제가 아니기 때문에 부팅을 한다고 해도 앱이 업로드 되어있지 않으면, 모니터/키보드/마우스가 연결되어 있다 하더라도 당장 크게 할 수 있는 일은 없습니다. 다만 안드로이드 앱을 업로드하기 위해서는 안드로이드 씽스가 PC와 같은 네트워크에 이더넷이나 WiFi로 연결되어 있어야 하는데 다음 장에서 이 부분을 다루도록 하겠습니다.

할 수 있는 부분이 없더라도 시스템 이미지가 제대로 설치되었는지 확인하기 위해 부팅을 해보겠습니다. 아래는 부팅 중 그리고 부팅 후의 안드로이드 씽스의 스크린 모습입니다.





연결되어 있는 네트워크가 없기 때문에 Not connected라고 뜹니다. 다음 장에서 네트워크에 연결시키면 이 부분에 연결된 네트워크의 종류와 안드로이드 씽스에게 할당된 IP주소가 표시되며, 이 IP주소를 이용해 PC에서 안드로이드 씽스에 접속할 수 있습니다.

4 안드로이드 씽스 시작하기

안드로이드 씽스는 안드로이드 씽스를 개발하는 PC와 같은 네트워크에 연결하는 것부터 시작합니다. 여기에는 아래와 같이 2가지 방법이 있습니다.

연결방법	준비물	소프트웨어
공유기로 연결	공유기/이더넷 케이블	없음
시리얼로 세팅	FTDI 시리얼 컨버터	Putty

안드로이드 씽스에 앱을 업로드하기 위해서는 먼저 안드로이드 씽스가 WiFi나 이더넷으로 연결된 후에 안드로이드 스튜디오에 내장되어 있는 Platform-Tools에 ADB(;Android Debug Bridge)를 이용하여 개발하는 PC와 안드로이드를 서로 연결되어 있어야 합니다. 스마트 폰의 경우 USB로 연결되어 있으면 바로 업로드 할 수 있지만, 아직 안드로이드 씽스는 그런 기능을 지원하지 않아 이런 방법을 거쳐야 합니다. 또한 스마트 폰 역시 앞서 설명한 방법을 통해 개발하는 PC와 네트워크로 연결해 놓을 수 있습니다.

먼저, 공유기를 통해 연결하면 그것 자체가 이더넷을 통해 네트워크에 연결된 것이므로, 별 다른 소프트웨어나 설정 없이 바로 ADB를 통해 연결하면 된다는 장점이 있지만, 스크린이 없는 경우, 안드로이드 씽스가 할당 받은 IP를 알아내기 위해 공유기에 접속하거나 IP를 스캔하는 등의 방법을 사용하여야 합니다.

시리얼을 통한 연결은 안드로이드 씽스와 시리얼 통신(Putty 이용)을 통해 WiFi에 접속하도록 한 후, 접속에 성공하면, 할당 받은 IP를 받아와서, 이 IP를 이용해 ADB로 연결하면 됩니다. 단점으로는 시리얼 연결을 하고, 약간의 명령어를 입력해야 하며, WiFi가 매번 잘 연결되지는 않는 다는 부분이 있으며, 장점으로는 스크린이 없을 때 시리얼 통신을 통해 안드로이드 씽스가 할당받은 IP를 알아내기 용이하다는 것입니다.

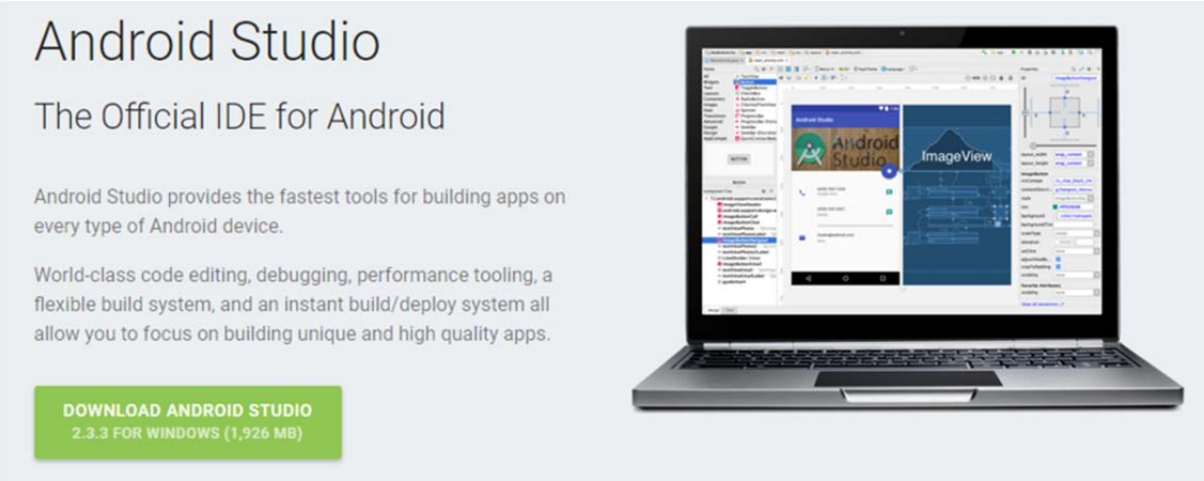
이렇게 연결하는 과정이 복잡한 것은 네트워크의 보안 문제이기 때문입니다. IP를 할당받고 포트를 여는 등의 과정이 없다면, 카페 등에서 다른 사람 스마트 폰에 접속하여 내부를 살펴보는 일 즉 해킹이 쉬워진다는 문제가 있습니다.

연결을 하기 전에 먼저 안드로이드 스튜디오를 설치하고 기타 설정을 하도록 하겠습니다.

4.1 안드로이드 스튜디오 설치하기

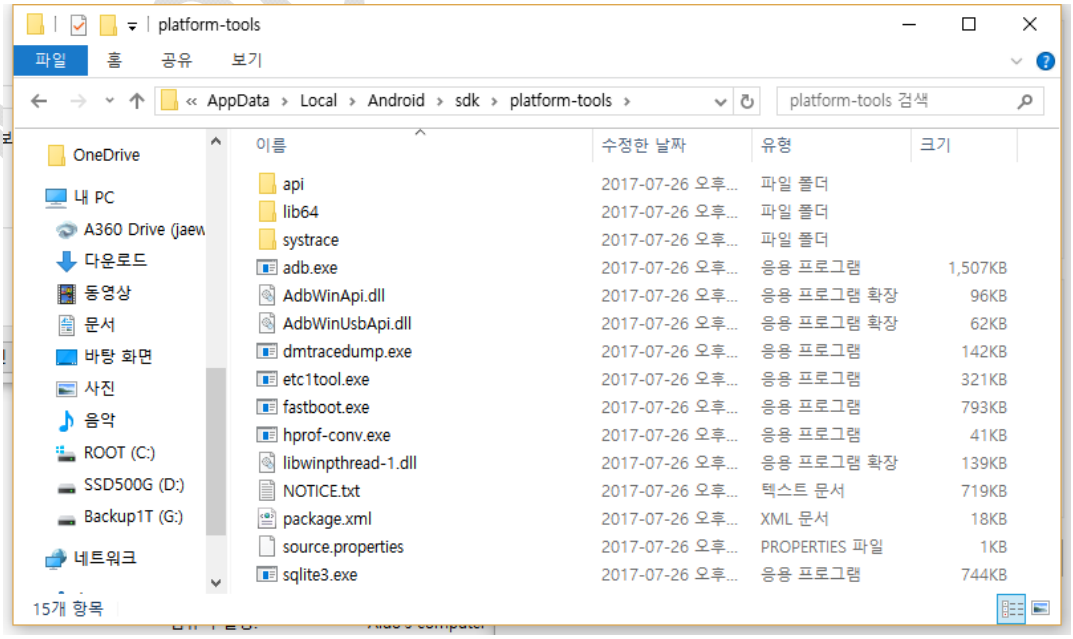
안드로이드 스튜디오 홈페이지에 가서 2.3 이상 버전의 안드로이드 스튜디오를 다운로드 받고 완료되면 설치합니다.

https://developer.android.com/studio/index.html



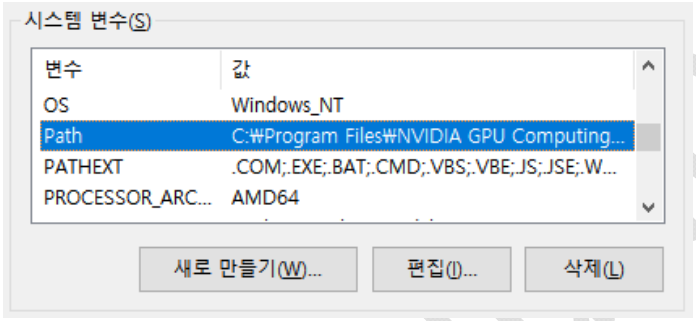
앞서 말한 네트워크에 연결된 안드로이드에 접속하려면 Platform-Tools 안에 있는 ADB가 필요합니다. 둘 다 안드로이드 스튜디오 안에 내장되어 있으나, ADB는 커맨드 라인 툴(프롬프트나 파워쉘에서 사용하는 프로그램)이기 때문에 편하게 사용하려면 ADB가 있는 폴더의 위치를 환경 변수에 저장해야 합니다. 먼저 이 폴더의 위치는 일반적으로 아래와 같습니다.

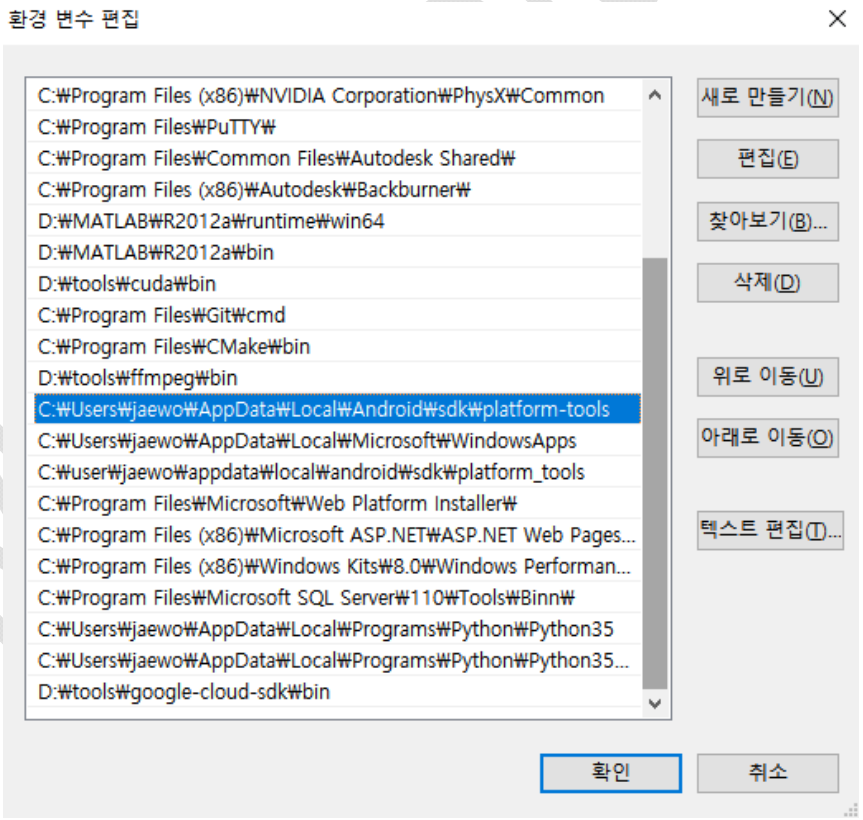
C:₩User₩<사용자 이름>₩AppData₩Local₩Android₩sdk₩platform-tools



<사용자 이름> 부분은 여러분들이 사용하는 윈도우즈를 켰을 때, 로그인 할 때 사용하는 ID를 입력하시면 되는데, 우선은 탐색기에서 위 폴더의 주소를 확실히 확인해보시기 바랍니다. 폴더를 잘 찾아갔다면 위와 같은 화면을 확인해보실 수 있습니다.

환경 변수를 설정하려면, 윈도우즈 10의 경우 시작메뉴 검색에서 "환경 변수"를 입력하거나 제어판에서 [제어판]-[시스템 및 보안]-[시스템]-[고급 시스템 설정]으로 간 다음 우측 하단에 [환경 변수]로 갑니다. 중앙의 시스템 변수에서 아래와 같이 Path를 찾아서 선택한 후 더블클릭을 하거나 편집을 누릅니다.





위와 같이 앞서 찾은 폴더 주소를 추가해 줍니다.

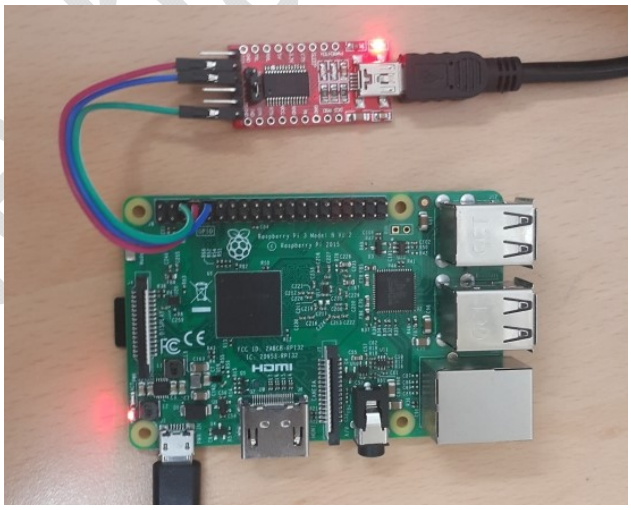
4.2 시리얼로 세팅하기

먼저 시리얼로 세팅하는 방법을 알아보겠습니다. 우선 FTDI라는 USB to 시리얼 컨버터가 필요합니다. 여기서 중요한 개념이 Logic 레벨이라 하여 대 HIGH의 기준이 3.3V인 것과 5V인 것이 있습니다. 5V이면 정확히 5V일 수가 없으니까, 보통 3.5V 이상이면 HIGH이다 라고 인식하여 3.3V가 들어오면 HIGH도 LOW도 아닌 것이 되어 통신이 안됩니다. 통신이 안될 뿐더러 3.3V 쪽에서는 5V의 전압과 전류를 못 버텨서 칩이 타버릴 수도 있습니다.

라즈베리파이의 GPIO는 3.3V이기 때문에 절대로 5V를 직접 연결하면 안되는데, 어째서인지 아두이노에 내장된 시리얼 컨버터를 이용하여 연결을 하였을 때 통신도 되고 라즈베리파이가 타버리지도 않았습니다. 결론적인 것만 말씀드리면, **절대 라즈베리파이의 GPIO에 5V를 입력하면 안됩니다.** 반드시 3.3V 시리얼 통신을 하셔야됩니다. 하지만 장시간 대용량 통신을 하지 않고, 초기 세팅에만 사용한다면 크게 문제가 되지는 않는다라고 말씀드리겠습니다. 아두이노는 있는데, 3.3V 컨버터가 없고, 굳이 3.3.3V 컨버터를 구입하는데 돈과 시간을 소모하고 싶으시지 않다면 **스스로 책임**을 지시고 아두이노를 사용하시면 됩니다. 혼란의 여지가 있기 때문에, 아두이노와 연결된 사진은 올리지 않도록 하겠습니다.

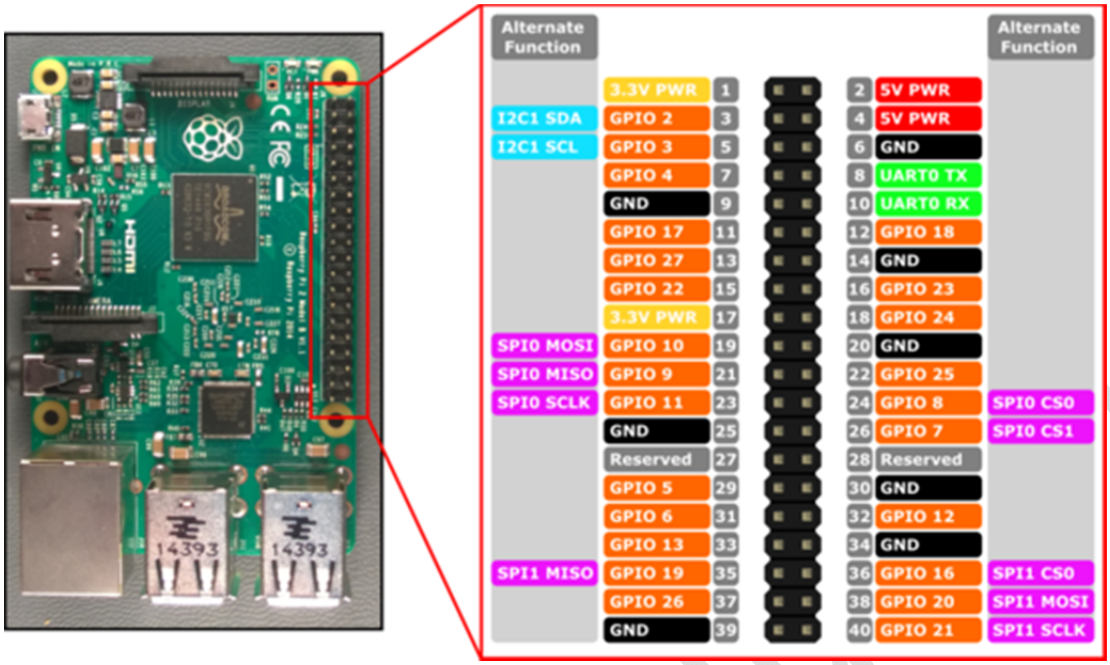
4.2.1 라즈베리파이와 시리얼 컨버터의 연결

SD카드를 라즈베리파이에 연결하고, 시리얼 컨버터와 라즈베리파이를 아래와 같이 연결합니다.



시리얼 포트를 연결할 때, GND는 GND, 전원이 필요할 경우 VCC는 VCC에 연결(3.3V인지 5V)인지 주의해서)하면 됩니다. 또한 송신과 수신이 TX, RX를 혼동하기 쉽습니다. 라즈베리파이의 TX는 라즈베리파이가 신호를 보내는 선이니, 컨버터의 RX에 연결하여야 합니다. 마찬가지로 라즈베리파이의 RX는 라즈베리파이가 신호를 받는 선이니, 컨버터의 TX에 연결하여야 합니다. 라즈베리파

이의 GPIO 배치는 아래와 같습니다.



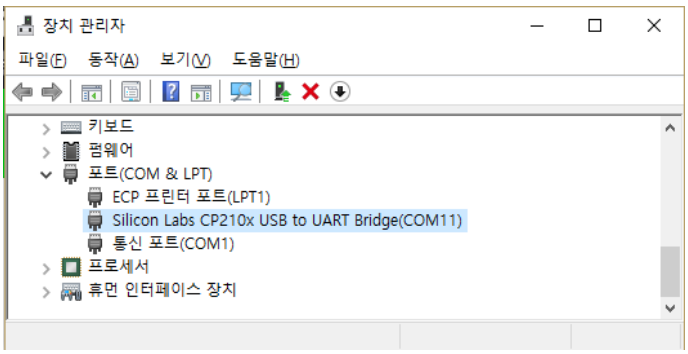
오른쪽 위부분부터 6이 GND, 8이 TX, 10이 RX입니다. 아두이노의 경우, 아두이노의 모서리에 표시되어 있는 TX, RX는 아두이노 기준이고, 아두이노의 컨버터 기준은 반대입니다. 그러므로 아두이노를 시리얼 컨버터로 이용하고자 할 때는 TX는 TX에, RX는 RX에 연결하여야 합니다.

4.2.2 시리얼 통신 연결

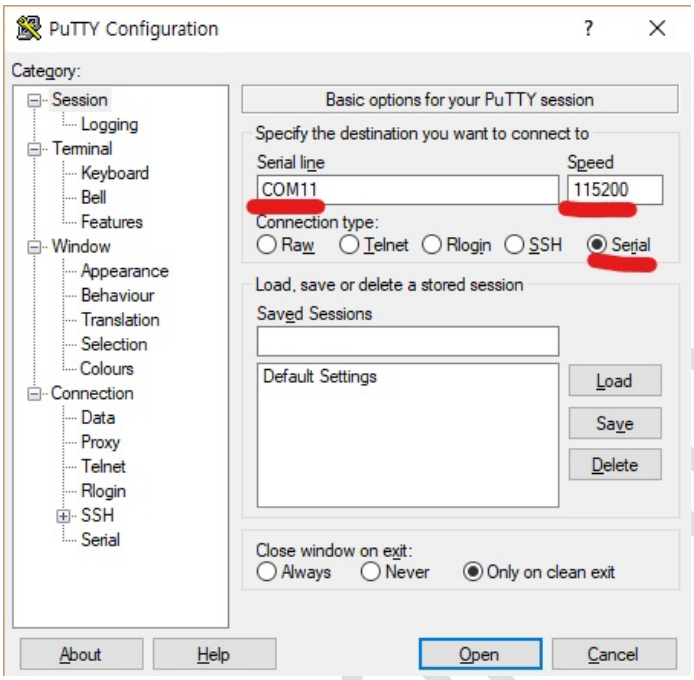
PC로 돌아와서 시리얼 통신 연결을 합니다. 시리얼 통신을 하는데에는 PUTTY라는 소프트웨어가 필요합니다. 아래 링크로 가서 프로그램을 다운로드 받습니다.

https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html

다운로드가 완료된 후에 설치를 하고 PUTTY를 실행합니다. 시리얼 통신을 연결하려면 먼저 컴퓨터에 연결된 USB 시리얼 컨버터의 포트 번호를 알아야 합니다. 아래와 같이 장치 관리자에서 포트 번호를 확인합니다.



포트 번호를 확인하였으니 PUTTY를 실행해서 아래와 같이 포트번호, 속도는 115200, 연결 타입은 시리얼로 설정, 입력합니다.



설정이 완료되었으면 Open을 눌러 연결한 후 라즈베리파이에 전원을 입력합니다. 연결이 제대로 되었다면 아래와 같은 화면이 뜹니다.

```
U-Boot 2017.01-g2ce116e560 (Jun 14 2017 - 03:23:10 +0000)

DRAM:  944 MiB
RPI 3 Model B (0xa22082)
MMC:   bcm2835_sdhci: 0
reading uboot.env
In:    serial
Out:   lcd
Err:   lcd
Net:   Net Initialization Skipped
No ethernet found.
ANDROID: Booting slot: a
ANDROID: reboot reason: "(none)"
Booting kernel at 0x1000000 with fdt at 2fffbc00...


## Booting Android Image at 0x01000000 ...
Kernel load addr 0x01000800 size 6963 KiB
RAM disk load addr 0x11000000 size 3580 KiB
## Flattened Device Tree blob at 2fffbc00
   Booting using the fdt blob at 0x2fffbc00
   XIP Kernel Image ... OK
   Loading Ramdisk to 3a7b7000, end 3ab35f82 ... OK
   Loading Device Tree to 2dff8000, end 2dfff38f ... OK
```

위는 부팅 초기입니다.

```
[    8.612615] init: Starting service 'bootanim'...
[    8.958624] type=1400 audit(8.529:12): avc: denied { open } for pid=156 comm=
"cameraserver" path="/dev" dev="tmpfs" ino=8193 scontext=u:r:cameraserver:s0 tcc
ntext=u:object_r:device:s0 tclass=dir permissive=1
[    9.971908] type=1400 audit(9.919:13): avc: denied { execmem } for pid=234 cc
mm="BootAnimation" scontext=u:r:bootanim:s0 tcontext=u:r:bootanim:s0 tclass=proc
ess permissive=1
[    9.988631] type=1400 audit(9.919:14): avc: denied { execute } for pid=234 cc
mm="BootAnimation" path="/dev/ashmem" dev="tmpfs" ino=6710 scontext=u:r:bootanim
:s0 tcontext=u:object_r:ashmem_device:s0 tclass=chr_file permissive=1
[   10.610976] type=1400 audit(10.559:15): avc: denied { dac_override } for pid=
165 comm="peripheralman" capability=1 scontext=u:r:peripheralman:s0 tcontext=u:r
:peripheralman:s0 tclass=capability permissive=1
[   14.664842] type=1400 audit(14.619:16): avc: denied { sys_module } for pid=16
4 comm="netd" capability=16 scontext=u:r:netd:s0 tcontext=u:r:netd:s0 tclass=cap
ability permissive=1
[   15.854746] capability: warning: `main' uses 32-bit capabilities (legacy supp
ort in use)
[   22.283048] healthd: battery l=100 v=0 t=42.4 h=2 st=2 chg=a
[   24.937164] smsc95xx 1-1.1:1.0 eth0: hardware isn't capable of remote wakeup
[   24.945354] IPv6: ADDRCONF(NETDEV_UP): eth0: link is not ready
[   28.309801] init: Service 'bootanim' (pid 205) exited with status 0

rpi3:/ $
```

위는 부팅이 끝난 후입니다. 이것을 adb shell이라 하며 여기서 안드로이드 쉘 명령어를 통해 안드로이드 씽스를 제어할 수 있습니다. WiFi 연결이 된 후에는 PUTTY가 없어도 프롬프트나 파워쉘에서 adb shell 명령으로 들어갈 수 있습니다.

4.2.3 WiFi 연결 설정하기

시리얼 통신이 연결된 후에는 다음과 같이 명령을 입력합니다.

```
am startservice ₩
-n com.google.wifisetup/.WifiSetupService ₩
-a WifiSetupService.Connect ₩
-e ssid <공유기 이름> ₩
-e passphrase <공유기 비밀번호>
```

₩ 없이 한줄로 입력해도 무관합니다. 명령어 am은 액티비티 매니저(Activity Manager)의 약자로 안드로이드 씽스에 기본적으로 내장되어 있는 wifisetup패키지 안에 있는 WiFiSetupService를 쉘 상에서 명령어로 제어하는 것입니다. 위 명령을 입력하면 아래와 같은 응답이 뜹니다.

```
etupService.Connect -e ssid aldonet -e passphrase aldoaldo                    <
Starting service: Intent { act=WifiSetupService.Connect cmp=com.google.wifisetup
/.WifiSetupService (has extras) }
```

WiFi에 연결을 합니다. 연결과정과 결과를 보려면 logcat이라는 명령어를 사용합니다.

logcat –d | grep wificonfigurator

logcat은 시스템에 보고된 에러 등의 기록(log)을 보는 명령이고 grep은 검색기능으로 wifi가 들어가있는 log만 출력하도록 하는 명령입니다. Logcat 뒤에 –d는 전체로그를 화면에 출력하고 종료하는 옵션입니다. –d가 없으면 logcat이 종료되지 않고 계속해서 log를 출력합니다. 정리하면 지금까지의 log 중에 wifi가 들어있는 log만 출력하고 바로 종료하라는 명령입니다. 결과는 다음과 같습니다.

```
rpi3:/ $ logcat -d | grep WifiConfigurator
09-22 05:55:55.491  1812  1812 I WifiConfigurator: Disabling access point
09-22 05:55:55.491  1812  1812 E WifiConfigurator: Failed to disable access poin
t
09-22 05:55:55.491  1812  1812 I WifiConfigurator: Clearing configurations
09-22 05:55:55.600  1812  1812 I WifiConfigurator: Disconnecting from network
09-22 05:56:16.494  1812  1812 I WifiConfigurator: Connecting to network with SS
ID "aldonet"
09-22 05:56:16.502  1812  1845 D WifiConfigurator: Adding new network
09-22 05:56:22.960  1812  1845 I WifiConfigurator: Successfully connected to ald
onet
rpi3:/ $
```

WifiConfigurator: Successfully connected to <WiFi 이름>이 뜨면 연결이 잘 되었다는 뜻입니다. 스크린이 연결되어 있다면 다음과 같이 WiFi가 연결되었음을 알려줍니다.

스크린이 있다면 위와 같이 안드로이드 씽스가 네트워크에 접속하여 할당받은 IP를 확인할 수 있습니다. 스크린이 없다면 shell에서 아래 명령을 통해서 할당받은 IP를 확인할 수 있습니다.

su
ifconfig

```
127|rpi3:/ $ su
rpi3:/ # ifconfig
wlan0     Link encap:Ethernet  HWaddr b8:27:eb:ea:31:ca  Driver brcmfmac_sdio
          inet addr:192.168.0.3  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::ba27:ebff:feea:31ca/64 Scope: Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:4760 errors:0 dropped:59 overruns:0 frame:0
          TX packets:882 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1336963 TX bytes:213294
```

4.2.4Adb 연결하기

안드로이드 씽스를 네트워크에 연결하였으니 이제 PC와 안드로이드 씽스를 연결할 일만 남았습니다. 프롬프트나 파워쉘을 실행합니다. 4.1 안드로이드 스튜디오 설치하기에서 platform-tools의 adb에 대한 환경변수 설정을 올바르게 했다면, adb 명령을 실행했을 때 아래와 같은 화면이 뜹니다.

```
PS C:₩> adb
Android Debug Bridge version 1.0.39
Revision 3db08f2c6889-android
Installed as C:₩Users₩jaewo₩AppData₩Local₩Android₩sdk₩platform-tools₩adb.exe

global options:
 -a         listen on all network interfaces, not just localhost
 -d         use USB device (error if multiple devices connected)
 -e         use TCP/IP device (error if multiple TCP/IP devices available)
 -s SERIAL
     use device with given serial number (overrides $ANDROID_SERIAL)
 -p PRODUCT
     name or path ('angler'/'out/target/product/angler');
     default $ANDROID_PRODUCT_OUT
 -H         name of adb server host [default=localhost]
 -P         port of adb server [default=5037]
 -L SOCKET  listen on given socket for adb server [default=tcp:localhost:5037]

general commands:
 devices [-l]             list connected devices (-l for long output)
 help                     show this help message
```

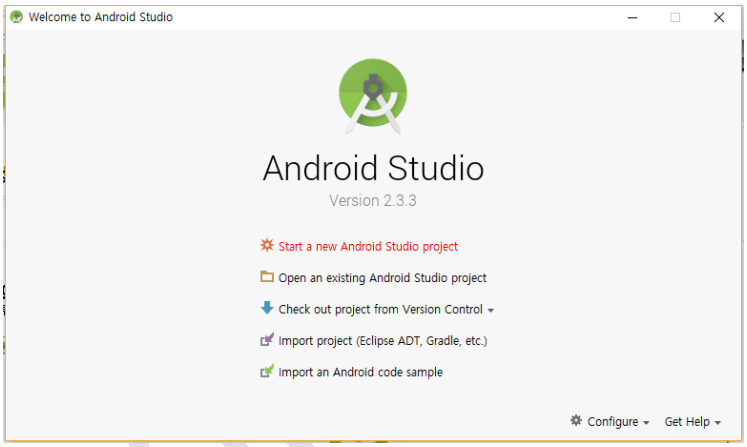
다음 명령을 통해 pc에서 안드로이드 씽스에 접속합니다.

adb connect <IP 주소>:5555

5555는 포트 숫자로 IP주소에 접속할 때 필요한 것입니다. 접속에 성공하면 다음과 같은 화면을 볼 수 있습니다.

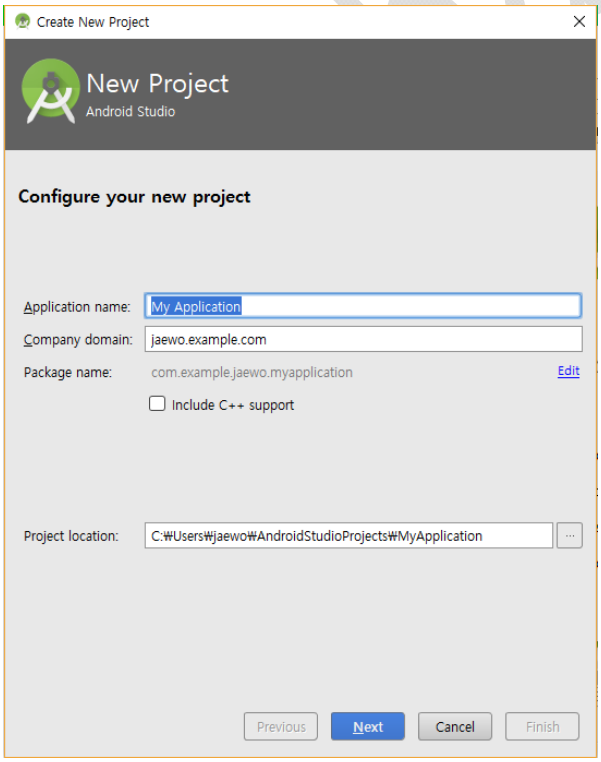
이제 안드로이드 스튜디오를 통해 앱을 개발하고 안드로이드 씽스에 업로드할 준비가 완료되었습니다.

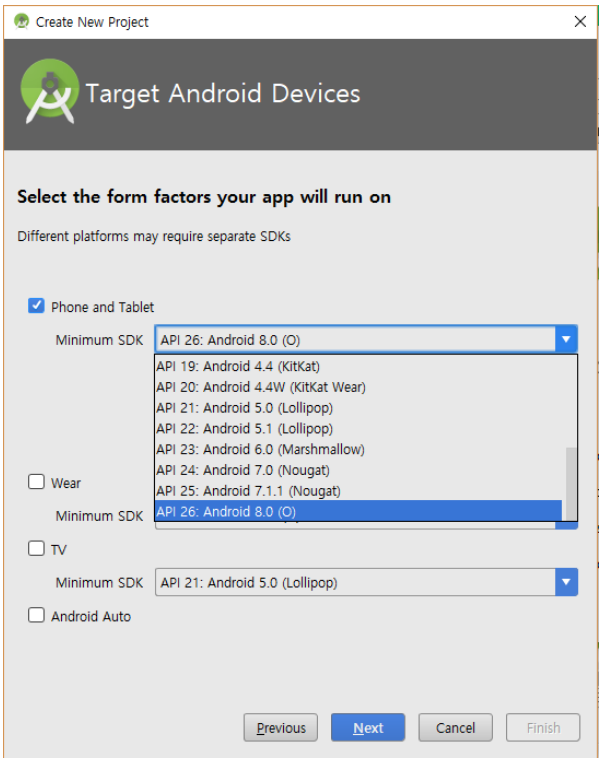
4.3 Hello World



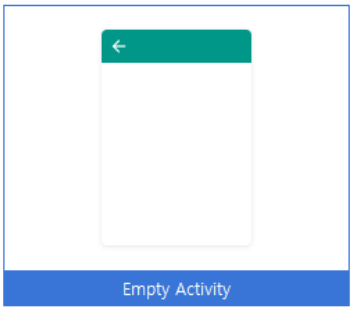
어떤 프로그래밍 언어든 개발 환경 설정이 끝나면 Hello World를 프린트함으로써 개발 환경이 잘 작동하는지 확인하는 과정을 거칩니다. 설치한 안드로이드 스튜디오를 실행합니다. 아래 화면에서 Start a new Android Studio project를 누릅니다.

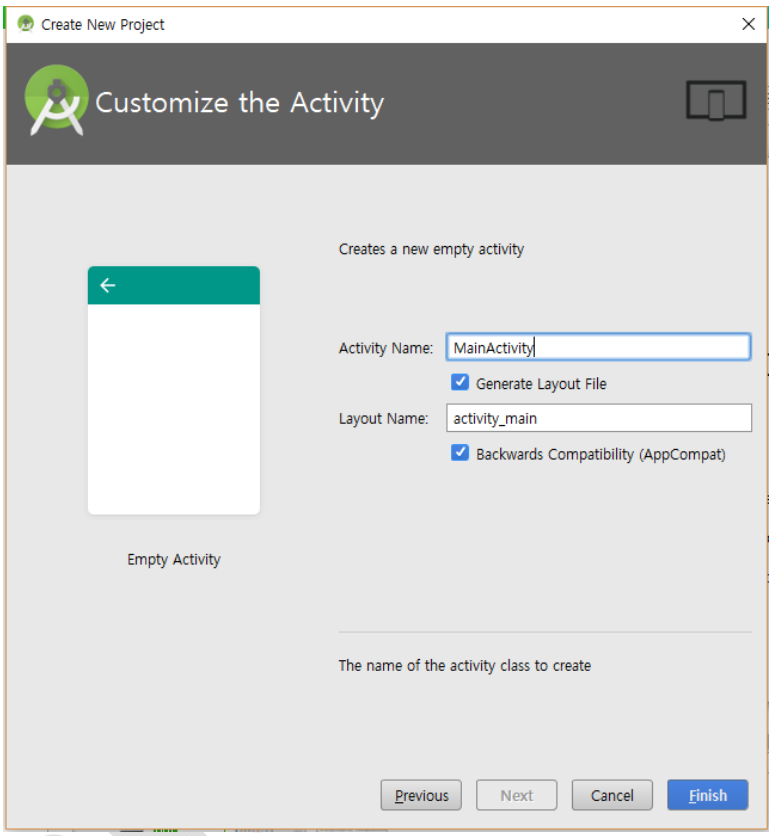
프로젝트 이름을 입력하고 Next를 누릅니다. 프로젝트가 저장되는 AndroidSudioProjects 폴더의 바로가기를 만들어 놓으면 추후에 프로젝트를 삭제하거나 옮기거나 할 때 편리합니다. 안드로이드 싱스를 위한 Minimum SDK는 API 25:Android 7.1.1 이상이어야 합니다.





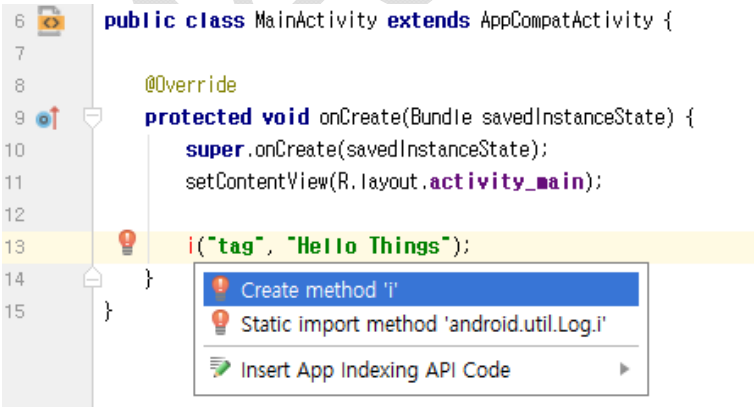
Activity 이름을 입력합니다. Finish를 눌러 프로젝트를 생성합니다.





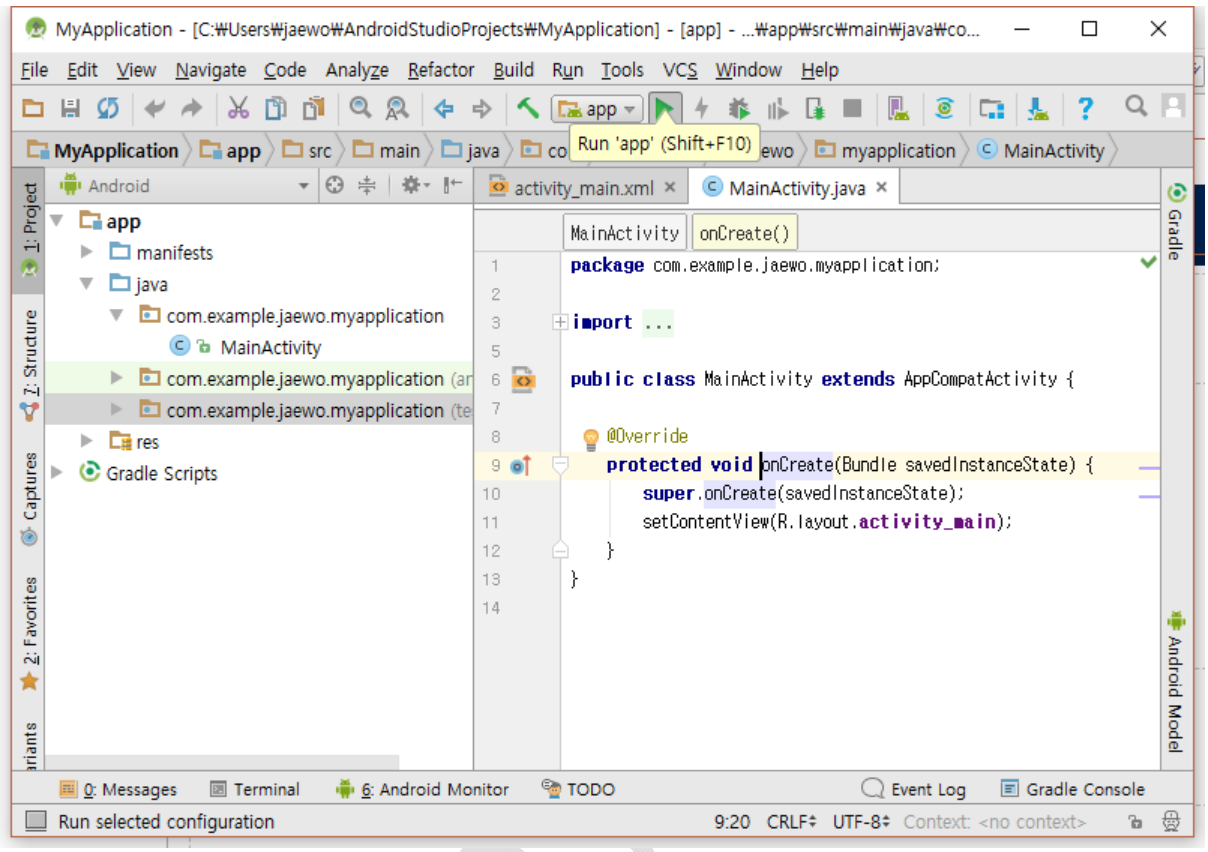
MainActivity 파일에서 setContentView.... 아래 줄에 아래와 같이 코드 한줄을 추가합니다.

i("tag", "Hello World"); 또는 android.util.Log.i("tag", "Hello World");

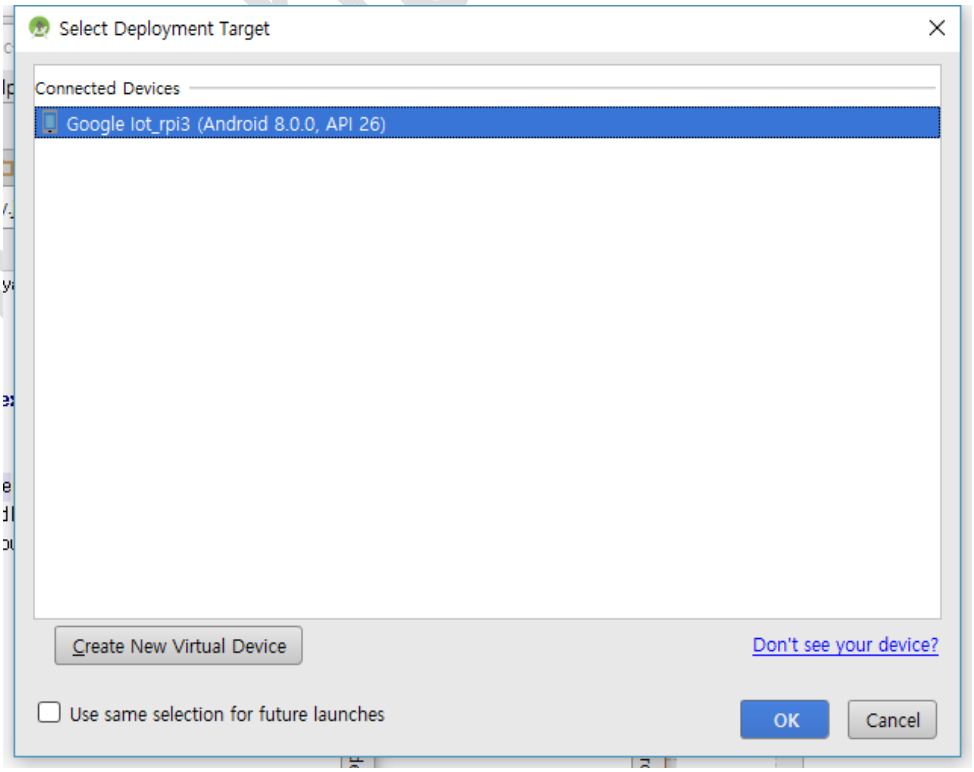


i 함수 부분에 빨간줄로 에러가 표시되는데, 커서를 i 함수에 놓고 Alt+Enter를 누르면 위와 같이 미리 선언되거나 Import 되지 않은 함수를 어떻게 할 것인지 물어보는 창이 뜹니다. 여기서 Static import method 'android.util.Log.i' 를 선택하시면 됩니다.

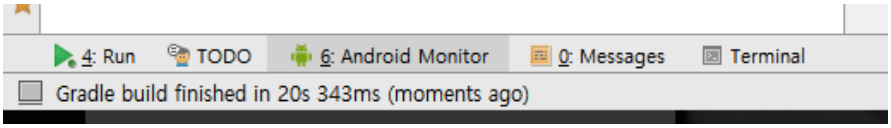
Import를 한 후 아래와 같이 툴바에서 Run을 눌러줍니다.



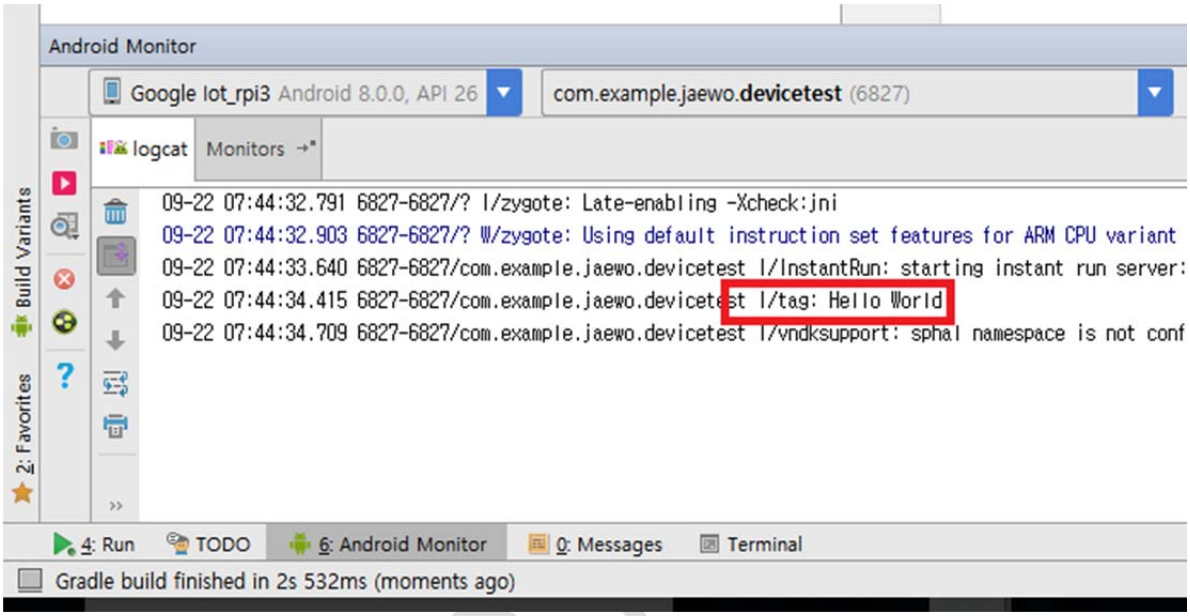
안드로이드 씽스가 adb를 통해 잘 연결되어 있다면 아래와 같은 화면을 볼 수 있습니다.



Run 후, 아래와 같이 안드로이드 스튜디오 좌측 하단에 Android Monitor를 클릭합니다.



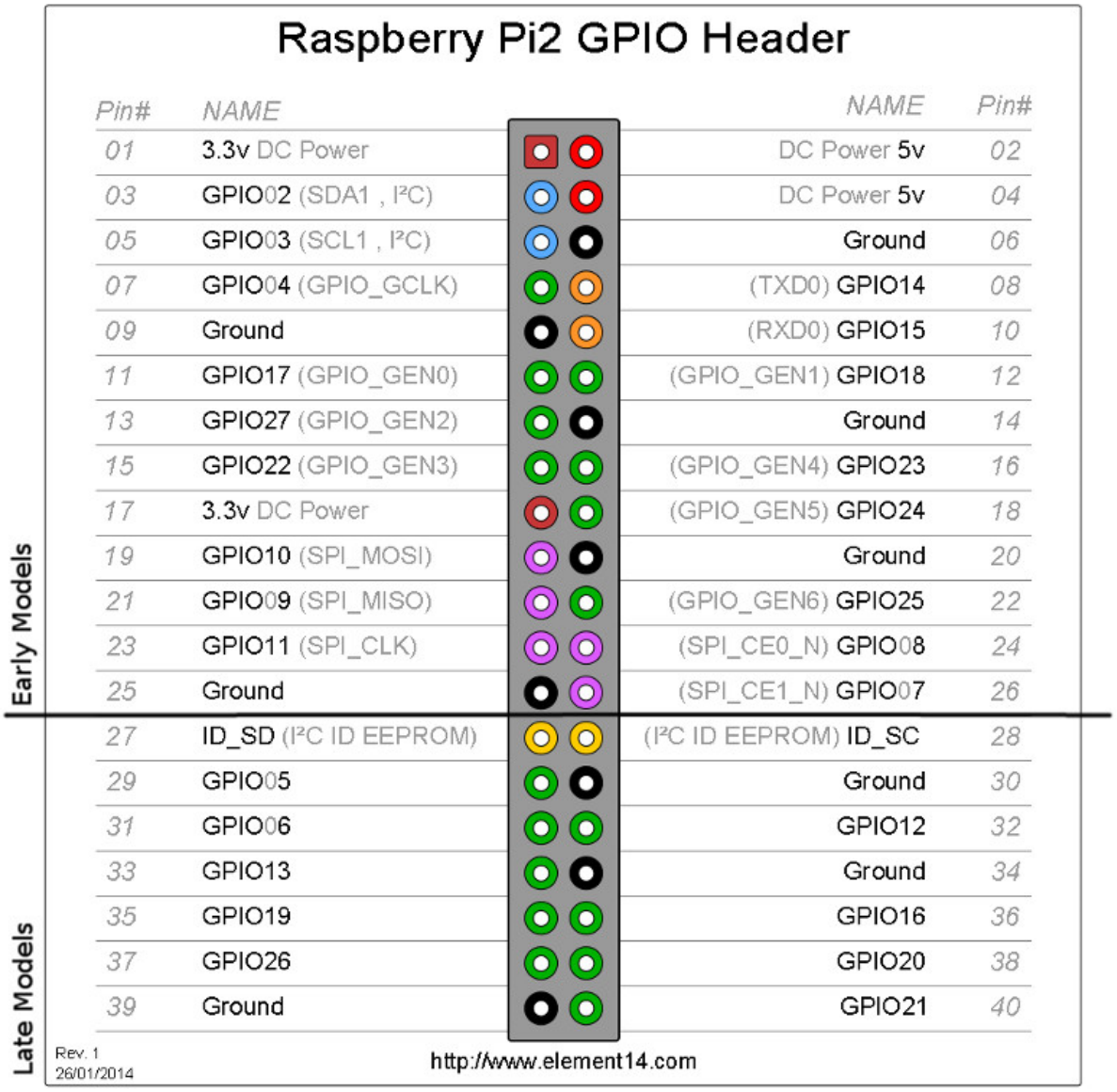
아래와 같이 logcat에 Hello World가 뜬 것을 확인할 수 있습니다.



앞서 입력한 i 함수는 android.util.Log 패키지에 내장된 함수로 이 함수를 사용하려면 android.util.Log 패키지를 Import하여야 합니다. android.util.Log 패키지는 log에 문자열(String)을 남기는 함수들을 제공하는데 이 log는 프로그램 상에서 사용자에게 보여지지는 않지만, 이렇게 개발하는 과정에서 개발자가 코드가 잘 작동하는지 여부를 확인할 때 사용합니다.

5 안드로이드 씽스 만들기

안드로이드 씽스에 앱을 업로드할 준비가 완료되었으니 이제 GPIO를 통해 입출력을 주고 받는 기본적인 안드로이드 씽스 앱을 개발해보도록 하겠습니다. GPIO는 라즈베리파이의 입출력 핀으로써 초기 세팅할 때 사용한 시리얼 UART를 포함해 SPI, I2C, I2S, PWM을 지원합니다. GPIO 핀의 배치는 아래와 같습니다.

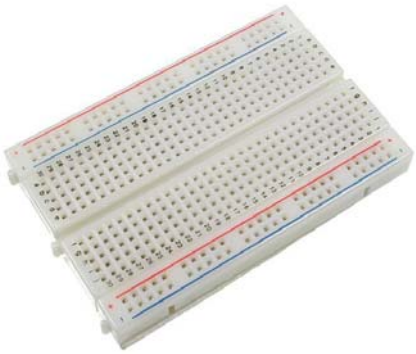


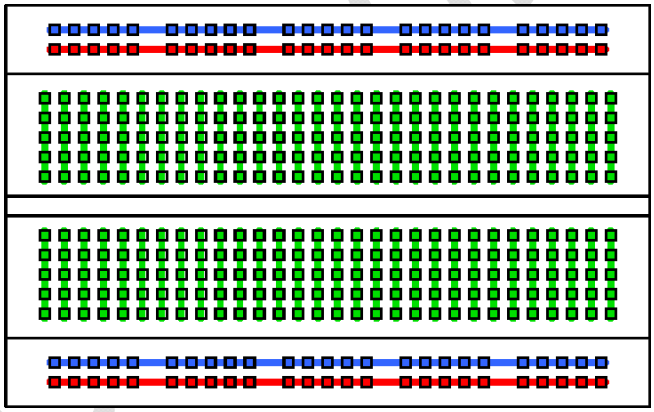
아날로그 입력을 받아 전압의 높낮이를 읽을 수 있는 아두니오와 달리, 라즈베리파이에는 ADC가 내장되어 있지 않습니다. 그러므로 아날로그 출력을 갖는 센서를 사용하기 위해서는 MCP3008 같은 별도의 ADC를 사용해야 합니다. 나머지 GPIO0~27은 아두이노의 디지털 입출력핀과 같은

역할을 합니다. 주의할 점은 주변장치에 전원을 공급하기 위해 5V 출력이 있기는 하나 라즈베리파이 내부에서는 3.3V를 사용하기 때문에 GPIO에 5V 입력을 가하면 프로세서가 파손될 수 있다는 것입니다. 입력을 받고자 하는 장치가 5V일 때는 3.3V-5V 로직 컨버터를 사용해야 라즈베리파이가 파손되는 것을 막을 수 있습니다.

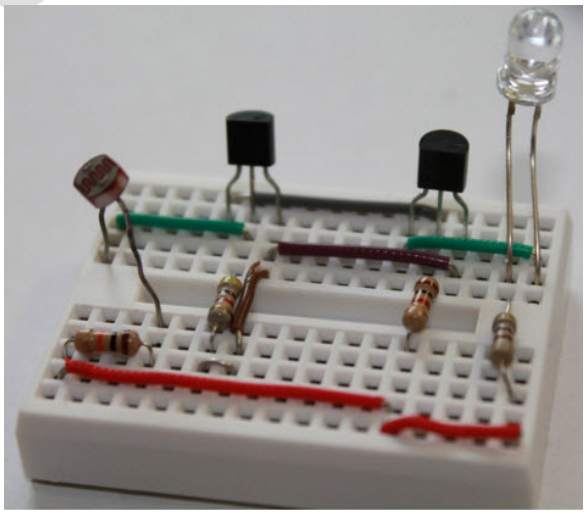
5.1 브레드 보드

안드로이드 씽스 만들기를 시작하기 앞서 전자회로의 기초이자 필수품이라고 할 수 있는 브레드 보드에 대해 알아보겠습니다. 소위 빵판이라고 불리는 브레드 보드는 아래와 필라스틱 판에 전자부품이나 점퍼선을 꽂을 수 있도록 구멍이 뚫려 있는 형태입니다.



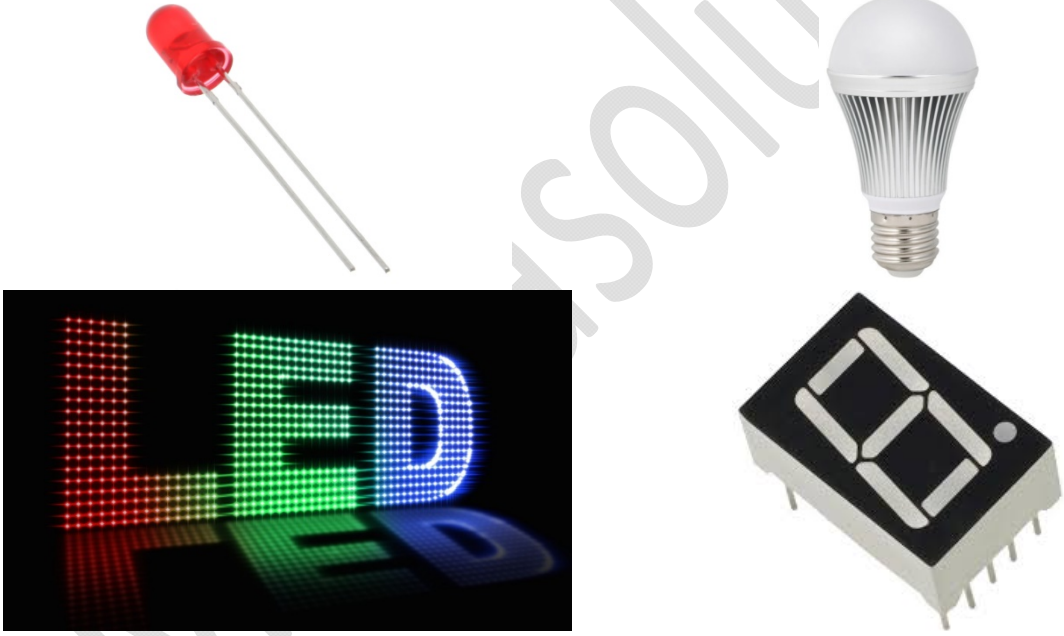


브레드 보드는 전자회로 구성 시에 회로를 테스트해보기 위한 목적으로 사용됩니다. 내부에 금속판이 있어 구멍에 점퍼선이나 전자부품을 꽂으면 위 오른쪽과 같은 형태로 서로 연결되게 됩니다. 여러가지 형태가 있지만, 대개 가운데 있는 구멍들은 위 오른쪽은 녹색선처럼 5개씩 연결되어 있고, 여기에 추가로 전원 연결을 원활히하기 위해 위 아래로 빨간색, 파란색 선을 따라 길게 연결되어 있는 전원부가 장착되기도 합니다. 아래는 브레드 보드에 회로를 구성한 예입니다.



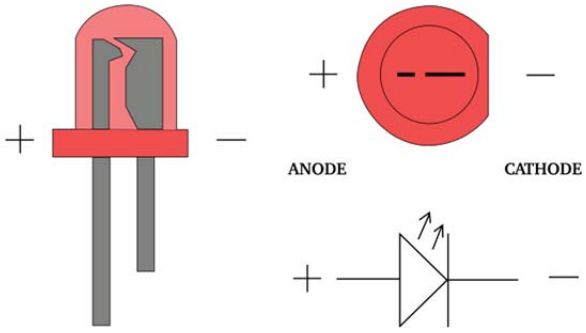
5.2 LED 제어하기

프로그래밍의 시작이 Hello World라면, 전자회로의 시작은 LED라고 할 수 있습니다. LED는 발광 다이오드(;Light Emiting Diode)의 약자로 전압이 가해지면 빛이 나는 전자 소자입니다. 일상에서 접하는 많은 전자제품들이 전원의 On/Off 여부나 작동상태를 표시하기 위해 LED를 사용합니다. 라즈베리파이에도 LED가 달려 있어서 전원이 들어오면 빨간 LED가 켜지는데, 이를 통해 라즈베리파이가 켜져있는지 여부를 확인할 수 있습니다. 고휘도LED를 사용하면 단순한 불빛 뿐 아니라, 조명도 가능하며, 빨강, 녹색, 파랑의 조합을 이용하여 다양한 색을 표현할 수도 있습니다. 최근 에너지 절약과 환경 보호가 화두가 되면서 과거 형광 등이나 전구 등을 수명이 길고 효율이 높은 LED로 대체하고 있기도 합니다. 또한 여러 개의 LED로 배열을 만들면 숫자나 글자를 표현할 수도 있습니다. 과거 네온사인을 대체하고 있는 LED 전광등이 그 예입니다. 이러한 배열을 하나의 부품으로 만든 세븐 세그먼트나 도트 매트릭스 등도 있습니다.



5.2.1 LED 이해하기

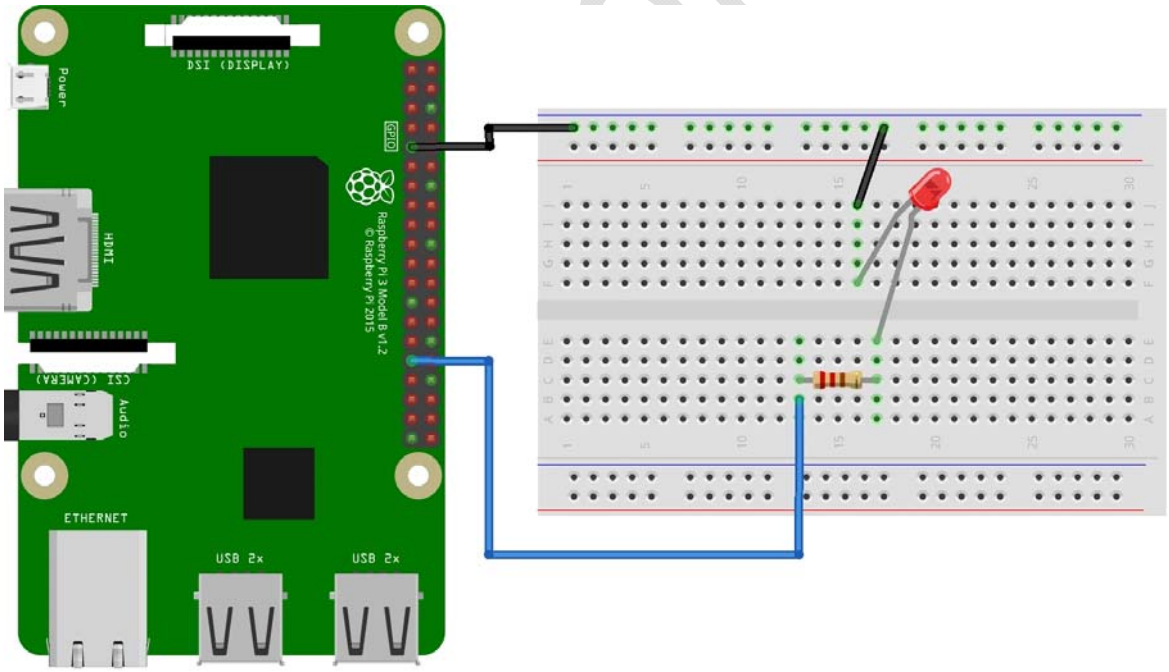
아래외 같이 회로를 구성합니다. LED는 2V 이상의 전압이 가해지면 파손될 수 있으므로 보통 저항과 직렬 연결하여 사용합니다. LED는 기본적으로 다이오드(Diode)라는 소자의 일종으로 전류를 한쪽 방향으로만 흘려보내는 특성이 있어서 반대방향으로 연결하면 불빛이 나오지 않게 됩니다. 극성을 쉽게 양극(+), 음극(-)라고 하고, 반도체 용어로 애노드(Anode, +), 캐소드(Cathode, -)라고 부르기도 합니다. LED의 극성 구별은 다음과 같이 합니다.



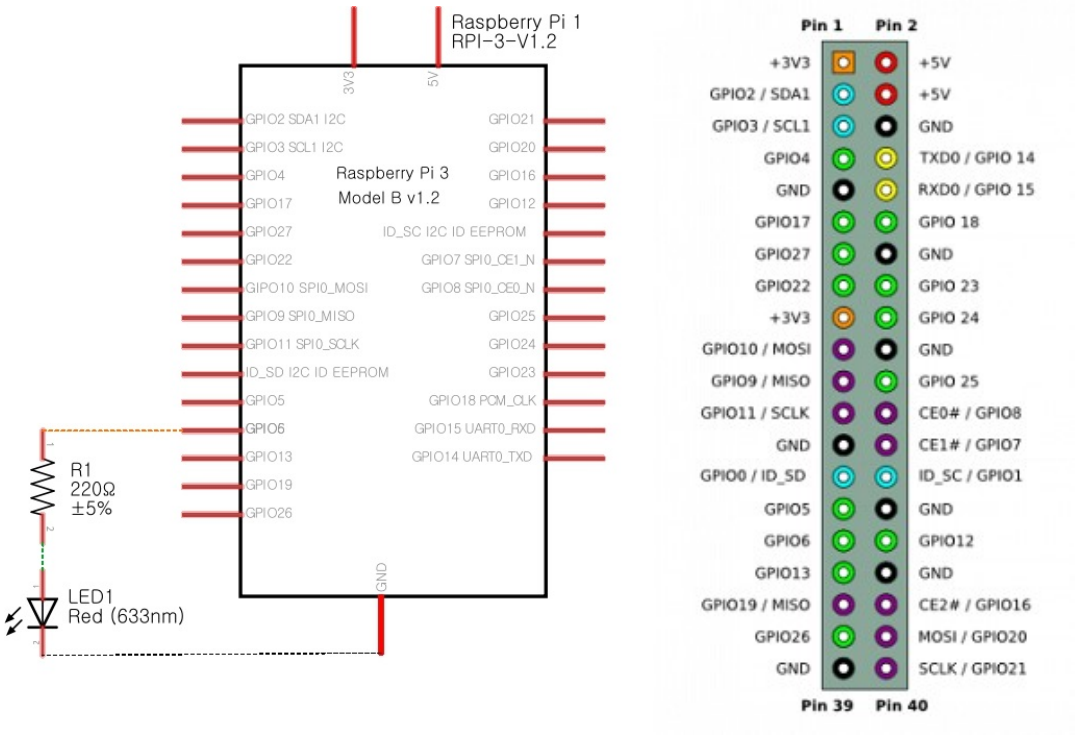
첫번째로 보는 것이 다리의 길이입니다. 길이가 긴쪽이 애노드고, 짧은 쪽이 캐소드입니다. 다리가 잘렸거나 해서 길이로 구별이 안되면, LED 내부를 봅니다. 내부에 금속이 작은 쪽이 애노드이고, 큰 쪽이 캐소드입니다. 앞서 설명드린 것 같이 다이오드는 전류가 애노드에서 캐소드로만 흐르며 그 반대방향으로는 흐르지 않으며 불도 켜지지 않습니다.

5.2.2 회로 구성하기

다음과 같이 브레드 보드에 회로를 구성합니다.



GPIO 핀 배치는 아래쪽과 같습니다. GPIO6에서 신호가 나와서 저항을 지난 다음 LED를 지나서 검은 선인 GND로 들어가게 됩니다. 그러므로 LED의 애노드 쪽이 저항과 연결되어야 합니다. 앞서 브레드 보드 부분에서 설명드린 것처럼 맨 위와 아래 부분은 빨간선과 파란선 표시대로 가로로 모두 연결되어 있으며, 중간 부분은 세로로 5개의 구멍이 서로 연결되어 있습니다. 회로도는 아래와 같습니다.



5.2.3 코드 작성하기

GPIO핀이나 기타 기능을 사용하기 위해서는 먼저 허가를 요청하거나 받고 관련 라이브러리를 불러와야 합니다. 이를 위해서 그래들(Gradle) 파일과 매니피스트(Manifest) 파일을 수정해줘야 합니다.

그래들이란?

그래들이란 그래들재단과 그래들웨어라는 업체가 개발한 오픈소스 빌드 도구입니다. 빌드란 앱이 만들어지기 전까지 필요한 소스들을 한 장소에 모으는 작업이라고 보면 됩니다. 안드로이드뿐 아니라 JAVA 프로젝트라면 무엇이든 사용되며, 프로그램의 테스트나 배포, 개발 등을 자동화할 수 있습니다. 안드로이드 스튜디오에 기본적으로 내장되어 있습니다.

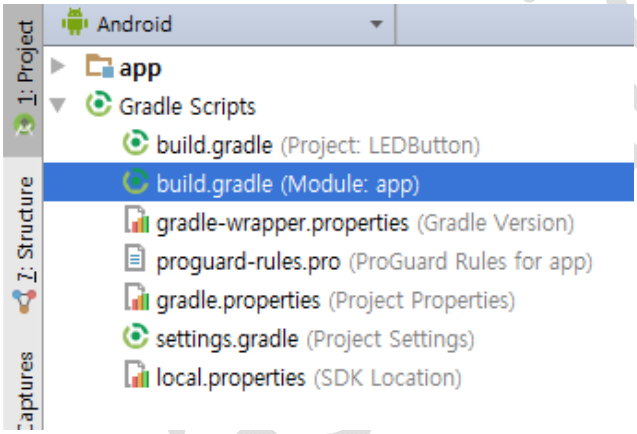
안드로이드 씽스에서의 그래들은, 아두이노와 비교할 수 있습니다. 아두이노 IDE의 툴 메뉴에서 우노인지 미니인지, 보드와 프로세서를 선택했던 것과 같습니다. 안드로이드의 경우 지원할 안드로이드의 버전이 몇인지? 어떤 라이브러리들을 사용할 것인지? 등등을 입력해주게 되는데, 이번의 경우 GPIO 같은 안드로이드 폰에는 없는 기능을 사용하기 위해 안드로이드 씽스 라이브러리를 불러와야 합니다.

매니페스트란?

매니페스트는 앱의 각종 정보를 저장하고 있는 신분증 같은 역할을 합니다. 앱을 위한 자바 패키지명, 앱을 구성하는 컴포넌트, 앱이 안드로이드의 제한된 API에 접근하거나 다른 앱의 제한된 컴포넌트를 사용하기 위한 권한을 설정할 수 있습니다. 블루투스 같은 각종 장치나 전화번호부 같은 기본 앱을 사용할 수 있도록 요청하는 부분이 여기 들어있습니다. 이번의 경우 안드로이드 씽스 라이브러리를 사용한다고 선언하는 부분이 들어가게 됩니다.

그래들 설정

먼저 그래들 파일을 열어줍니다.



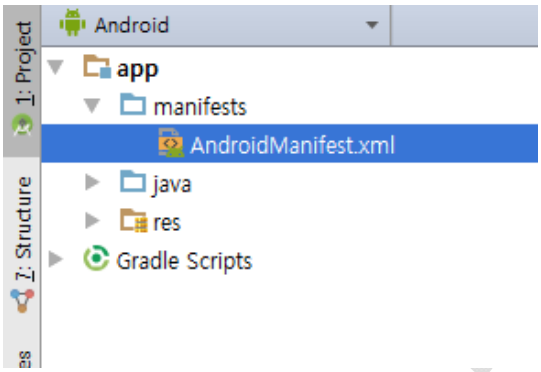
아래 쪽 dependencies에 다음 내용을 아래 그림과 같이 추가해줍니다. GPIO는 기존의 안드로이드에 포함되어 있는 내용이 아니기 때문에 Things Support Library를 추가해줘야 합니다.

```
dependencies {
    ...
    provided 'com.google.android.things:androidthings:0.5.1-devpreview'
}
```

```
21
22      dependencies {
23          compile fileTree(dir: 'libs', include: ['*.jar'])
24          androidTestCompile('com.android.support.test.espresso:espresso-core:2.2.2', {
25              exclude group: 'com.android.support', module: 'support-annotations'
26          })
27          compile 'com.android.support:appcompat-v7:26.+'
28          compile 'com.android.support.constraint:constraint-layout:1.0.2'
29          testCompile 'junit:junit:4.12'
30
31          provided 'com.google.android.things:androidthings:0.5.1-devpreview'
32      }
33
```

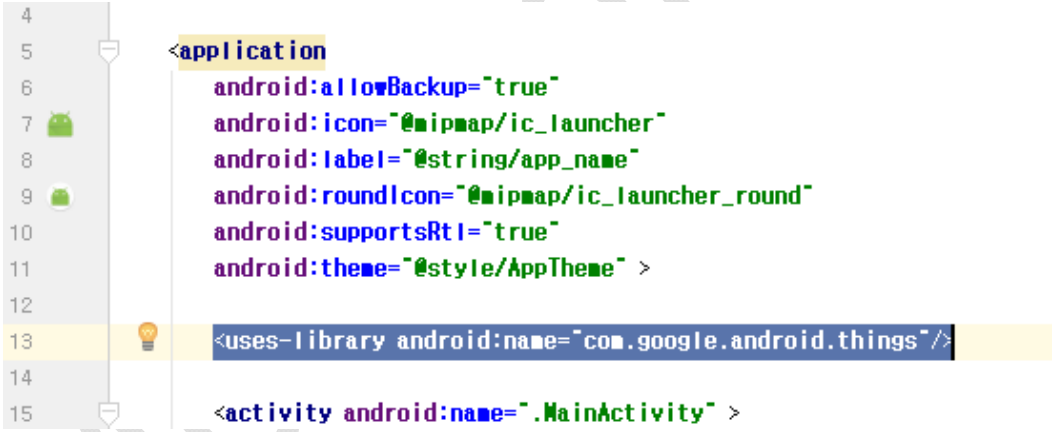
매니페스트 설정

수정된 그래들 파일을 저장한 후, 아래와 같이 매니페스트 파일을 열어줍니다.

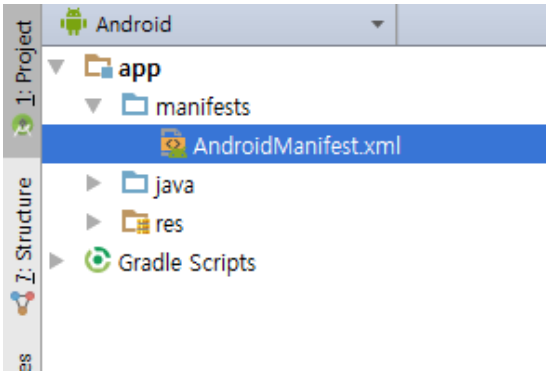


Things shared library를 아래와 같이 추가해 줍니다.

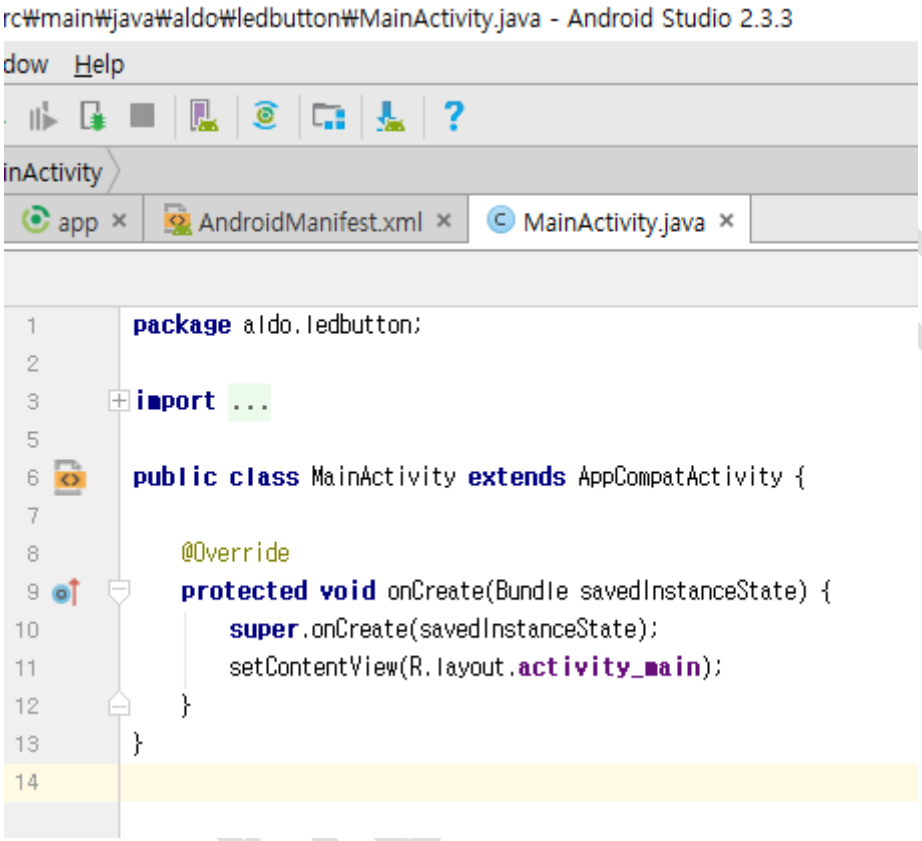
```
<application ...>
    <uses-library android:name="com.google.android.things"/>
    ...
</application>
```



이제 코드의 메인이 되는 MainActivity를 살펴보겠습니다. 아래와 같이 MainActivity 파일을 엽니다.



우선은 LED를 켜보는 것이 집중하여 코드를 살펴보도록 하겠습니다. 관련 지식이 없으신 분들은 지금 당장은 이해하기 어려울 수 있으나, 이해하지 못해도 무방하니 다음 순서로 넘어가시기 바랍니다.



먼저 6번 줄을 보면 public이라고 되어 있는데, 이는 말 그대로 공용 즉 누구나 접근 및 사용할 수 있음을 의미합니다. 안드로이드는 객체를 기본으로 구성됩니다. 객체 지향 프로그래밍(Object Oriented Programming)의 개념을 잘 모르시는 분들은 객체(Object)란 단순하게 변수와 함수 등을 묶어 놓은 개념이라고 생각하시면 됩니다. 이런 객체를 선언(Declaration)할 때 클래스(Class)를 씁니다.

모든 안드로이드의 기초가 되는 것이 Activity 클래스(AppCompatActivity는 안드로이드 4.0 이하 버전과의 호환성을 위한 것으로 Activity로 바꾸어도 무방합니다.)입니다. 그래서 안드로이드 SDK에서 기본적인 Activity 클래스에 앱에 필수로 있어야 하는 변수와 함수들을 정의해놓고, 각각의 개발자들이 자신들의 프로젝트를 생성할 때 이 Activity 클래스를 상속(extends) 받아 사용하도록 하고 있습니다. 여기서 상속이란 객체를 확장할 때 사용하는 말로 자식 객체가 부모 객체의 모든 변수와 함수 등을 모두 그대로 가져오고, 여기서 확장하여 자신 만의 변수와 함수를 추가하는 것을 말합니다. 여기서 자식 객체는 MainActivity, 부모 객체는 AppCompatActivity입니다.

6번 줄을 다시 해석하자면 객체 MainActivity를 선언하는데, AppCompatActivity를 부모 삼아 확

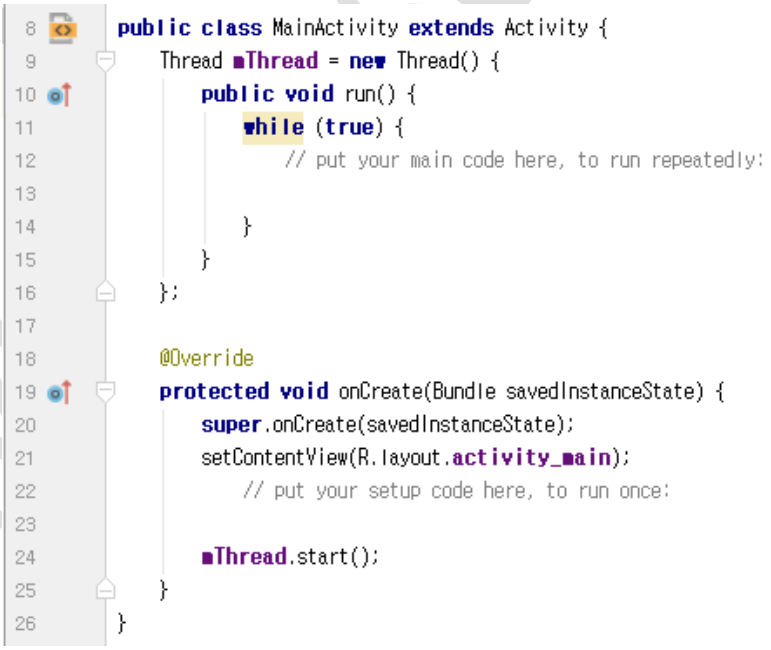
장(extends)하여 사용한다는 의미입니다. 8번 줄부터 보시면, Override라는 말이 나오는데 이는 onCreate함수가 부모 객체에 있는 함수인데, 이를 상속/확장하여 사용한다는 의미입니다. 함수 내부를 보면 10번 줄에 super.onCreate(...);이 부모 객체에서 구현된 부분을 실행하는 함수입니다. 11번 줄의 setContentView 함수는 화면을 그려주는 함수입니다. 안드로이드 씽스에 스크린을 연결하지 않을 것이라면 11번 줄은 지워도 무방합ㄴ디ㅏ.

먼저 처음하시는 분들이 접근하기 쉽도록, 코드는 다소 어렵고 복잡하더라도 아두이노와 비슷한 형태가 되도록 만들어보겠습니다. 다음 코드를 아래 사진과 같은 위치에 붙여 넣으세요. 첫번째 코드 :

```
Thread mThread = new Thread() {
    public void run() {
        while (true) {
        // put your main code here, to run repeatedly:
        }
    }
};
```

그리고 두번째 코드 :

```
mThread.start();
```



첫번째 함수의 While 문 내부를 아두이노의 loop으로 생각하시면 되고, 두번째 함수 onCreate 중간에 빈 줄을 아두이노의 setup으로 생각하시면 됩니다. 7번줄에 나오는 쓰레드(Thread)란 하나의 작업을 의미합니다. 아두이노의 loop를 하나의 쓰레드라고 생각하시면 됩니다. 아두이노는 단 하나의 쓰레드만 가지고 실행할 수 있는 반면, 안드로이드나 일반 컴퓨터에서는 여러 개의 쓰레

드를 실행, 즉 한번에 여러가지 작업을 병행할 수 있습니다. 하지만 쓰레드가 하나도 없는 경우도 있기 때문에 지금처럼 주기적으로 LED를 켰다 껐다하는 작업을 위해서는 하나의 쓰레드를 작성해줘야 합니다.

먼저 LED를 컨트롤하고, 그 상태를 기억해놓기 위한 변수를 하나씩 만들어보겠습니다. 맨 위의 Main Activity와 Thread 사이에 넣어주시면 됩니다.

```
public class MainActivity extends Activity {
    private Gpio mLedGpio = null;
    private boolean mLedVal = true;

    Thread mThread;
    …
```

Gpio 클래스는 Gpio 핀들을 입출력으로 이용하는데 필요한 클래스입니다. 아두이노의 Digital 핀과 같다고 생각하시면 됩니다. 그 다음으로는 아래와 같이 핀들의 초기 설정을 해줍니다.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    PeripheralManagerService pioService = new PeripheralManagerService();
    try {
        mLedGpio = pioService.openGpio("BCM6");
        mLedGpio.setDirection(Gpio.DIRECTION_OUT_INITIALLY_LOW);
        mLedGpio.setValue(true);
    } catch (IOException e) {}
    mThread.start();
}
```

PeripheralManagerService 클래스는 GPIO 클래스에 지정된 핀 BCM6를 담당하는 객체를 생성해주는 역할을 합니다. try 문은 예외사항이 발생할 가능성, 즉 실행하는 과정에서 에러가 날 가능성이 있는 함수를 사용할 때 필수적으로 사용해야 하는 함수입니다. 만약 에러가 나면 뒤의 catch 문 안에 있는 명령을 실행하게 됩니다만 현재는 비어있습니다. 예외사항의 예로는 이 프로그램이 라즈베리파이가 아닌 다른 보드에 설치되었을 경우를 들 수 있습니다. BMC6는 라즈베리파이의 GPIO핀의 이름이기 때문에 다른 보드에서는 에러가 나게 됩니다.

try문 내부는 아두이노에서 pinMode처럼 생각하시면 됩니다. 핀을 출력으로 초기값은 LOW로

설정해줍니다. 아두이노와 다른 점은 GPIO핀을 초기화하는데 별개의 클래스, PeripheralManagerService,가 필요하다는 부분입니다. 이게 초기 설정을 마쳤으니 Loop 문으로 가보겠습니다.

```
Thread mThread = new Thread() {
    public void run() {
        while (true) {
        // put your main code here, to run repeatedly:
        if (mLedVal) {
            mLedGpio.setValue(false);
            mLedVal = false;
        } else {
            mLedGpio.setValue(true);
            mLedVal = true;
        }
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {}
    }
};
```

마찬가지로 try 문을 사용합니다. LED가 켜져 있으면 끄도록, 꺼져있으면 켜도록 코드를 작성했습니다. 마지막으로 Thread.sleep에서 1초(1000ms)를 기다리도록 했습니다. 아두이노의 delay함수가 같다고 생각하시면 됩니다. 전체 코드는 다음과 같습니다.

```
public class MainActivity extends Activity {
    private Gpio mLedGpio = null;
    private boolean mLedVal = true;
    Thread mThread = new Thread() {
        public void run() {
            while (true) {
            // put your main code here, to run repeatedly:
            if (mLedVal) {
                mLedGpio.setValue(false);
                mLedVal = false;
            } else {
                mLedGpio.setValue(true);
                mLedVal = true;
            }

            try {
```

```
                    Thread.sleep(1000);
                } catch (InterruptedException e) {}
            }
        };
        @Override
        protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.activity_main);

            PeripheralManagerService pioService = new PeripheralManagerService();
            try {
                mLedGpio = pioService.openGpio("BCM6");
                mLedGpio.setDirection(Gpio.DIRECTION_OUT_INITIALLY_LOW);
                mLedGpio.setValue(true);
            } catch (IOException e) {}

            mThread.start();
        }
}
```

이제 프로그램을 업로드하면 LED가 1초 간격으로 깜빡거리는 것을 확인하실 수 있습니다.

블로그 자료 : http://blog.naver.com/roboholic84/221094025910

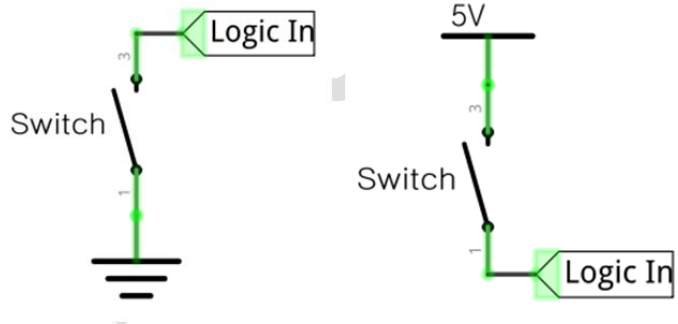
5.3 버튼 입력 받기

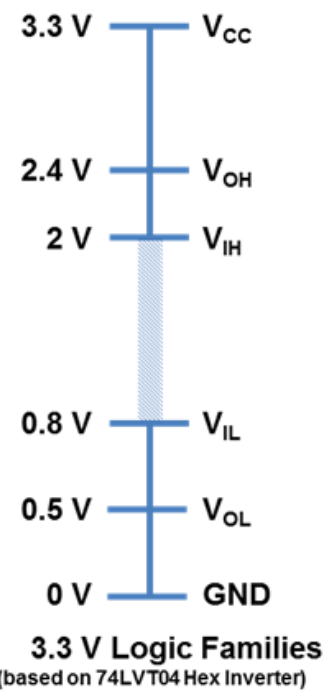
버튼 입력은 아두이노에서도 다소 난이도가 있는 코딩입니다. 버튼이 눌린 시점 또는 떼진 시점에서 특정 코드를 동작시키려면 인터럽트(Interrupt)라는 개념이 필요하기 때문입니다. 인터럽트란 특정 핀에 함수를 연결해주는 것으로 문자 그대로 현재 진행 중인 일을 멈추고 인터럽트 함수 먼저 실행하는 것입니다. 버튼이나 기타 On/Off 형태의 센서 출력을 받을 때 사용하며, 가장 흔한 예가 엔코더 값을 읽는 것입니다. 또한 실시간(Real Time) 시스템을 만들 때 내부 클럭에 물려서 사용하기도 합니다. 인터럽트 함수를 인터럽트가 걸렸을 때 부른다 하여 콜백함수(CallBack Function)이라고 부릅니다.

안드로이드에서는 주로 터치 스크린 입력을 처리하는 기능을 지원하고, 여기에 더하여 앱 간이나 앱 내부에서 인텐트를 통해 컴포넌트 간에 데이터를 보낼 때도 인터럽트가 발생합니다. 다른 점이라면 GPIO를 통해 버튼의 On/Off를 받는다는 것 때문에 별도의 드라이버를 지정해 줄 필요가 있습니다.

5.3.1 풀 업/다운 이해하기

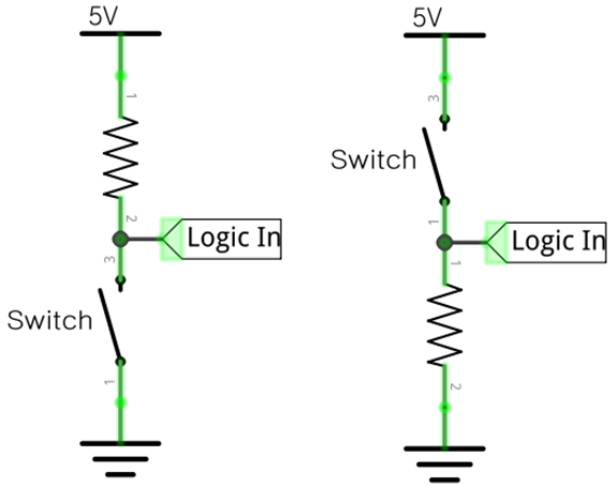
회로 상에 버튼을 구현하거나 기타 물리적으로 전선의 연결과 제거가 일어날 때, 풀 업/다운을 반드시 해주어야 합니다. 풀 업이란, 스위치가 눌렸을 때 0V가 연결되게 한다면 떨어졌을 때는 5V가 되도록 만들어주는 것이고, 풀 다운이란 스위치가 눌렸을 때 5V가 연결되게 하였다면 떨어졌을 때 0V가 되게 만들어 주는 것입니다. 아래 도면을 보겠습니다.





각각 스위치가 눌리면 0V, 5V가 입력되도록 하였습니다. 스위치가 떨어져 있을 때 입력 핀의 전압은 몇 V 일까요? 실제는 중간 전압에서 왔다갔다합니다. 이 상태를 떠있는 상태(Floating)이라고 합니다. 오른쪽 그림을 보면 라즈베리파이가 몇 V 전압을 기준으로 HIGH/LOW를 결정하는지 나타나 있습니다. 2V 이상을 HIGH, 0.8V 이하를 LOW로 인식하는데, 앞에서 처럼 핀이 떠있게

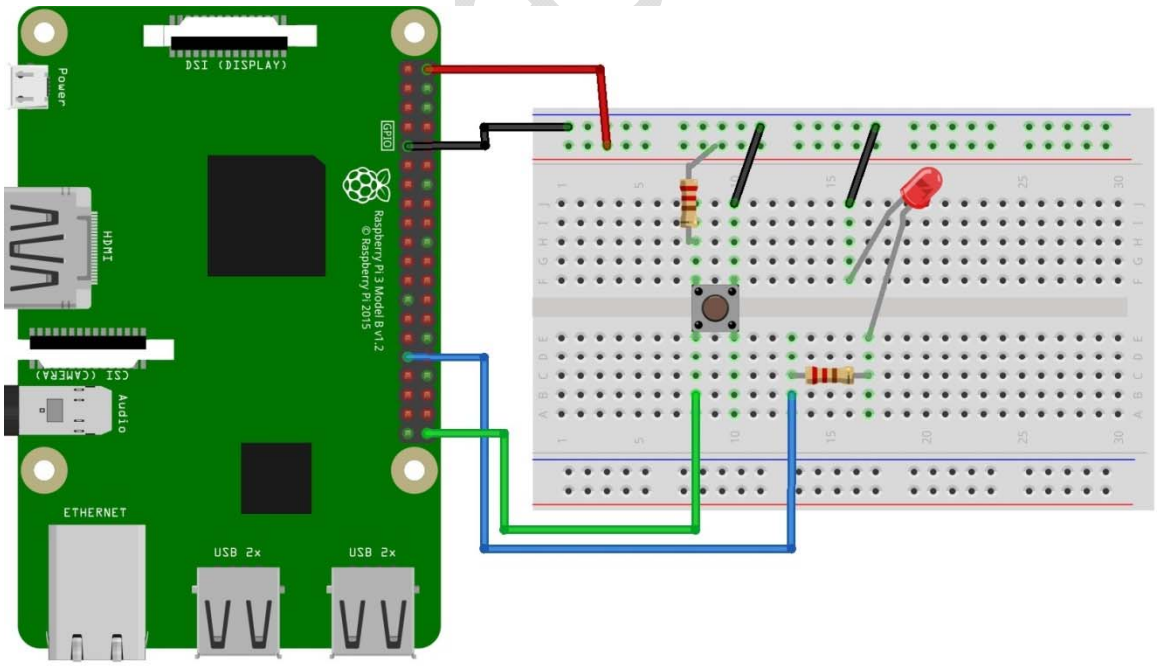
되면 라즈베리파이는 HIGH와 LOW를 왔다갔다 하는 것처럼 인식합니다. 이런 문제를 해결하기 위해 풀업/다운 저항을 달아주게 됩니다.



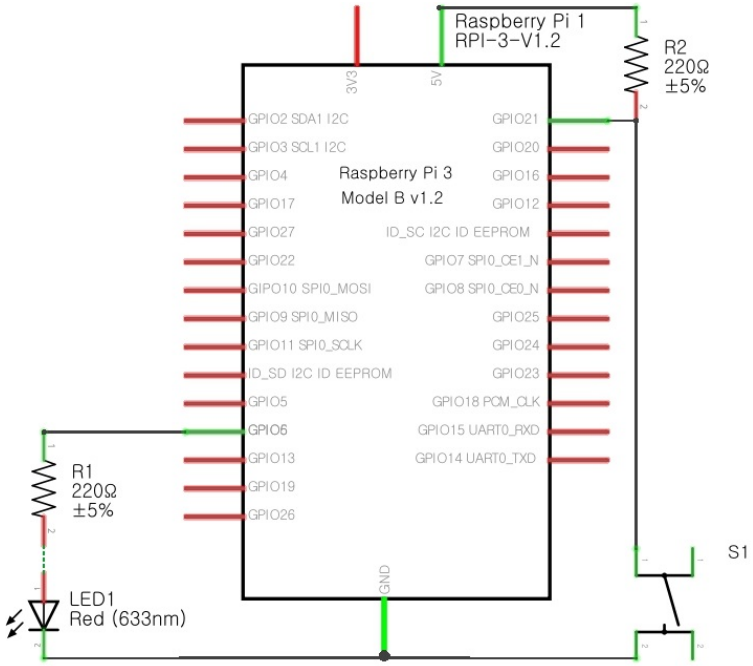
위와 같이 풀업 풀 다운 저항을 달아주면 스위치가 떨어졌을 때는 각각 HIGH, LOW가 되고, 스위치가 눌렸을 때는 각각 LOW, HIGH가 되게 됩니다.

5.3.2 회로 구성하기

다음과 같이 브레드 보드에 회로를 구성합니다.



4핀 푸쉬 버튼은 일반적으로 같은 방향을 보고 있는 핀끼리 연결되어 있고, 서로 다른 방향끼리는 스위치를 눌렀을 때만 연결이 됩니다.



5.3.3 코드 작성하기

그래들 설정

dependencies에 다음을 추가합니다. LED 켜고 끄기와 다르게 버튼 드라이버도 필요하게 됩니다.

```
dependencies {
    ...

    compile 'com.google.android.things.contrib:driver-button:0.4'
    provided 'com.google.android.things:androidthings:0.5.1-devpreview'
}
```

```
22      dependencies {
23          compile fileTree(dir: 'libs', include: ['*.jar'])
24          androidTestCompile('com.android.support.test.espresso:espresso-core:2.2.2', {
25              exclude group: 'com.android.support', module: 'support-annotations'
26          })
27          compile 'com.android.support:appcompat-v7:26.'
28          compile 'com.android.support.constraint:constraint-layout:1.0.2'
29          testCompile 'junit:junit:4.12'
30
31          compile 'com.google.android.things.contrib:driver-button:0.4'
32          provided 'com.google.android.things:androidthings:0.5.1-devpreview'
33      }
```

매니페스트 설정

매니페스트에 다음을 추가합니다. 먼저 버튼 드라이버를 사용하기 위해 승인을 요청합니다.

```
<manifest>
    <uses-permission android:name="com.google.android.things.permission.MANAGE_INPUT_DRIVERS" />

    ...
</manifest>
```

안드로이드 폰에서는 앱에서 요청을 할 때 사용자에게 허용할 것인지 물어보지만, 안드로이드 씽스에서는 스크린이나 기타 입출력 장치가 없는 경우가 있고, 애초에 장치 개발을 목적으로 하기 때문에 허용이 무의미해서 물어보지는 않습니다만, 앱이 작동하지는 않습니다. 승인이 필요한 경우 안드로이드 씽스에서는 재부팅을 하면 승인이 되어 있게 됩니다. 재부팅은 간단하게 라즈베리파이에 연결된 전원을 제거 후 재연결하거나, 프롬프트에서 adb reboot 명령을 입력하면 됩니다.

이전 포스트에서와 마찬가지로 안드로이드 씽스 라이브러리를 사용하겠다고 선언해줍니다.

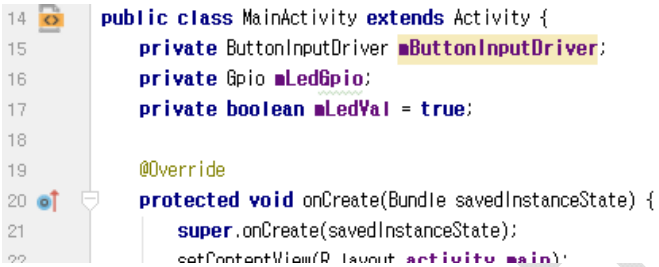
```
<application>
    <uses-library android:name="com.google.android.things"/>

    ...
</application>
```

```
2   <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3       package="aldo.ledbutton" >
4
5       <uses-permission android:name="com.google.android.things.permission.MANAGE_INPUT_DRIVERS" />
6
7       <application
8           android:allowBackup="true"
9           android:icon="@mipmap/ic_launcher"
10          android:label="LEDButton"
11          android:roundIcon="@mipmap/ic_launcher_round"
12          android:supportsRtl="true"
13          android:theme="@style/AppTheme" >
14
15          <uses-library android:name="com.google.android.things"/>
16
17          <activity android:name=".MainActivity" >
```

코드는 이전과 유사한 것이 있으나, 버튼 드라이버를 추가하는 것과, 쓰레드가 아닌 버튼이 눌리는 시점을 이벤트로 처리하는 키다운 이벤트 함수가 사용됩니다. 멤버 객체/변수는 이전의

Gpio 클래스의 LedGpio 객체와 부울 변수 LedVal만 사용하던 것에서 ButtonInputDriver 클래스의 객체가 추가됩니다.

```
private ButtonInputDriver mButtonInputDriver;
private Gpio mLedGpio;
private boolean mLedVal = true;
```



초기화는 마찬가지로 onCreate 함수에서 실행되며 이전 코드에서 ButtonInputDriver의 초기화가 추가됩니다. 초기화 값은 우선 버튼에 연결된 핀인 BCM21, 핀의 디지털 값이 LOW일 때가 버튼이 눌린 것으로 지정, 마지막으로 버튼을 스페이스(키보드에서 스페이스)로 지정하였습니다. 만약 안드로이드 씽스에 키보드가 연결되어 있다면 스페이스 키를 눌렀을 때 버튼을 누른 것으로 인식하게 됩니다. 마지막으로 레지스터 함수를 통해 등록을 합니다.

```
PeripheralManagerService pioService = new PeripheralManagerService();

try {
    mLedGpio = pioService.openGpio("BCM6");
    mLedGpio.setDirection(Gpio.DIRECTION_OUT_INITIALLY_LOW);
    mLedGpio.setValue(true);
} catch (IOException e) {}

try {
    mButtonInputDriver = new ButtonInputDriver(
        "BCM21",
        Button.LogicState.PRESSED_WHEN_LOW,
        KeyEvent.KEYCODE_SPACE);
        mButtonInputDriver.register();
} catch (IOException e) {}
```

```
19      @Override
20      protected void onCreate(Bundle savedInstanceState) {
21          super.onCreate(savedInstanceState);
22          setContentView(R.layout.activity_main);
23
24          PeripheralManagerService pioService = new PeripheralManagerService();
25
26          try {
27              mLedGpio = pioService.openGpio("BCM6");
28              mLedGpio.setDirection(Gpio.DIRECTION_OUT_INITIALLY_LOW);
29              mLedGpio.setValue(true);
30          } catch (IOException e) {}
31
32          try {
33              mButtonInputDriver = new ButtonInputDriver(
34                      "BCM21",
35                      Button.LogicState.PRESSED_WHEN_LOW,
36                      KeyEvent.KEYCODE_SPACE);
37              mButtonInputDriver.register();
38          } catch (IOException e) {}
39      }
```

버튼이나 스크린 터치는 필수적으로 사용되는 기능이기 때문에 Activity 클래스에 내장이 되어 있습니다. 여러분들이 할 일은 이 내장된 함수 onKeyDown을 상속 받아서 필요한 기능만 넣는 것입니다. 우선 이 onKeyDown 함수는 어떤 버튼이든 시스템에 등록된 버튼이 눌리거나 떼었을 때 실행되며, 어떤 키인지는 함수의 매개변수인 keyCode를 통해서 알 수 있습니다. LED를 제어하는 함수는 이전 포스트의 그것을 그대로 이용하였습니다.

```
@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
    if (keyCode == KeyEvent.KEYCODE_SPACE) {
        try {
            if (mLedVal) {
                mLedGpio.setValue(false);
                mLedVal = false;
            } else {
                mLedGpio.setValue(true);
                mLedVal = true;
            }
        } catch (IOException e) {}
        return true;
    }

    return super.onKeyDown(keyCode, event);
}
```

```
41      @Override
42      public boolean onKeyDown(int keyCode, KeyEvent event) {
43          if (keyCode == KeyEvent.KEYCODE_SPACE) {
44              try {
45                  if (mLedVal) {
46                      mLedGpio.setValue(false);
47                      mLedVal = false;
48                  } else {
49                      mLedGpio.setValue(true);
50                      mLedVal = true;
51                  }
52              } catch (IOException e) {}
53              return true;
54          }
55
56          return super.onKeyDown(keyCode, event);
57      }
```

전체 코드는 다음과 같습니다.

```
import android.app.Activity;
import android.os.Bundle;
import android.view.KeyEvent;

import com.google.android.things.contrib.driver.button.Button;
import com.google.android.things.contrib.driver.button.ButtonInputDriver;
import com.google.android.things.pio.Gpio;
import com.google.android.things.pio.PeripheralManagerService;

import java.io.IOException;

public class MainActivity extends Activity {
    private ButtonInputDriver mButtonInputDriver;
    private Gpio mLedGpio;
    private boolean mLedVal = true;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        PeripheralManagerService pioService = new PeripheralManagerService();

        try {
            mLedGpio = pioService.openGpio("BCM6");
            mLedGpio.setDirection(Gpio.DIRECTION_OUT_INITIALLY_LOW);
            mLedGpio.setValue(true);
        } catch (IOException e) {}
```

```
        try {
            mButtonInputDriver = new ButtonInputDriver(
                    "BCM21",
                    Button.LogicState.PRESSED_WHEN_LOW,
                    KeyEvent.KEYCODE_SPACE);
            mButtonInputDriver.register();
        } catch (IOException e) {}
    }

    @Override
    public boolean onKeyDown(int keyCode, KeyEvent event) {
        if (keyCode == KeyEvent.KEYCODE_SPACE) {
            try {
                if (mLedVal) {
                    mLedGpio.setValue(false);
                    mLedVal = false;
                } else {
                    mLedGpio.setValue(true);
                    mLedVal = true;
                }
            } catch (IOException e) {}
            return true;
        }

        return super.onKeyDown(keyCode, event);
    }
}
```

블로그에 방문하시면 작동 동영상을 확인할 수 있습니다.

http://blog.naver.com/roboholic84/221095347538

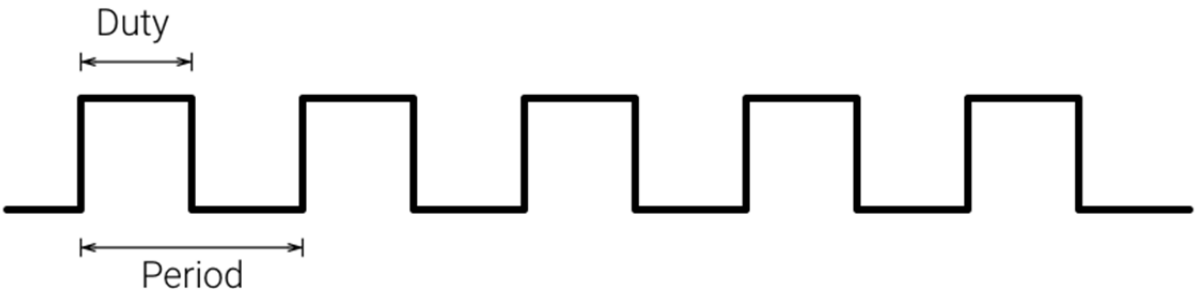
5.4 서보 모터 제어하기

이번 장에서는 안드로이드 씽스(Android Things)로 서보 모터를 제어하는 예제 프로그램을 만들어보겠습니다. 서보 모터는 원래 특정 모터를 지칭하는 것이 아니라, 어떤 모터에 제어 시스템이 갖춰진 것을 말하는 것이며, 그것이 계속 사용되다 보니 마치 모터의 종류인 것처럼 서보 모터라고 부르게 되었습니다. 서보 모터의 조건은 모터와 센서, 제어 시스템이 3가지가 되겠습니다. 크게 CNC 등 고속 정밀 제어를 요하는 산업현장에 사용되는 것이 있고, 우리가 일상에서 쉽게 접할 수 있는 RC용 또는 로봇용 서보모터가 있습니다. 이 역시 원래는 RC항공기 계열에서 주로 사용하다가, 로봇이 유행하게 되면서 메이커 쪽에서도 많이 사용하게 되었고, 로봇용 서보 모터도 나오게 되었습니다.

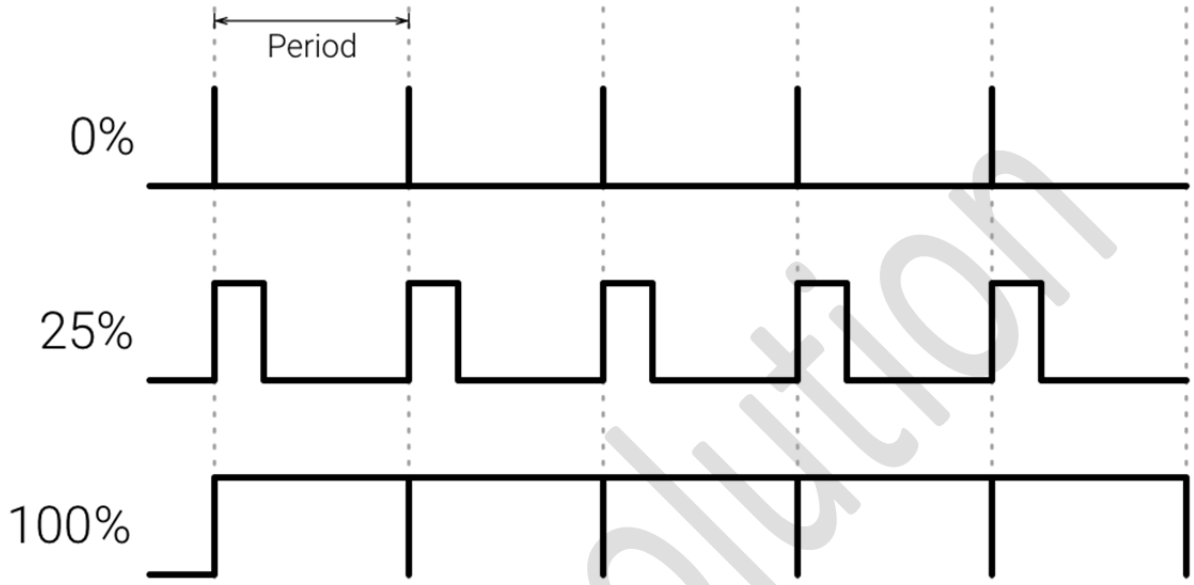
로봇용 서보 모터는 여러 편리한 기능을 제공하지만, 비교적 고가이므로, 아직까지는 RC 서보 모터를 많이 사용하고 있습니다. RC 서보 모터는 PWM 신호를 입력을 받아서 모터의 각도를 제어할 수 있는 서보 모터입니다. PWM은 대부분의 마이크로 컨트롤러에서 지원하고 있고, 그 위에 서보 모터 제어 라이브러리도 흔하게 찾을 수 있습니다.이번 장에서는 안드로이드 씽스에서 지원하는 서보 라이브러리를 이용해 서보모터를 한쪽 끝에서 다른쪽 끝까지 움직이는 앱을 만들어보도록 하겠습니다.

5.4.1 PWM 이해하기

PWM이란 Pulse Width Modulation의 약자로 진폭변조라는 뜻입니다.

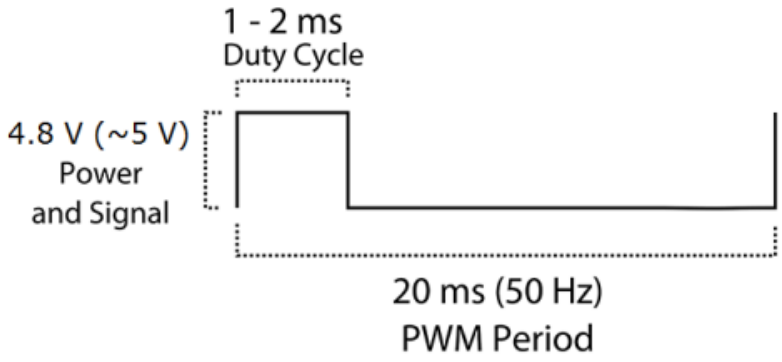


위와 같은 사각파에서 HIGH로 올라갔다가 LOW로 내려가는 것을 펄스(Pulse)라고 하고, 그 시간을 진폭(Pulse width 또는 Duty)이라고 합니다. 그리고 다음 번에 다시 HIGH로 올라가는 때까지 걸리는 시간을 주기(Period)라고 합니다. 주기의 역수 1/Period가 주파수가 됩니다. 여기서 PWM 신호는 아래와 같이 주기(Period)는 고정되어 있고, 진폭(Duty)를 변화시키는 것을 의미합니다.



PWM은 아날로그의 성격을 띄는 신호이면서 디지털 회로에서 만들기 쉽다는 장점 때문에, 매우 다양한 분야에서 사용되고 있습니다. 보기에는 디지털 신호이나, 넓은 범위에서 신호의 전력량으로 접근하면, 전압을 바꾸는 것과 비슷한 효과를 볼 수 있습니다. 가령 5V 25% PWM은 1.25V와 같다고 할 수 있습니다.

RC용 서보 모터 제어 시에는 PWM의 진폭이 모터의 각도와 대응되게 됩니다.

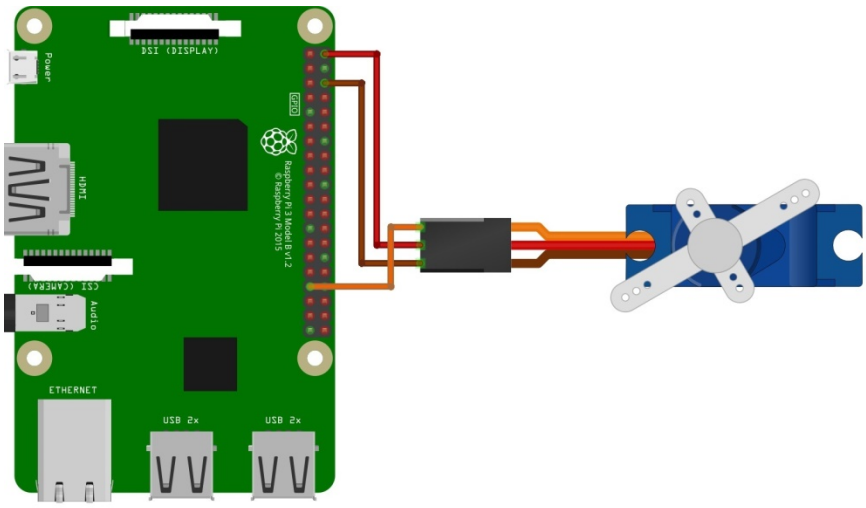


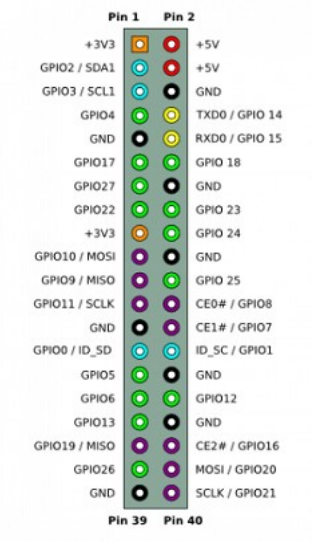
Position "0" (1.5 ms pulse) is middle, "90" (~2 ms pulse) is all the way to the right, "-90" (~1 ms pulse) is all the way to the left.

이번 장에서 사용할 SG90 서보 모터의 경우 20ms 주기(50Hz 주파수)에서 진폭이 1ms이면 -90도, 2ms이면 +90도가 되게 됩니다.

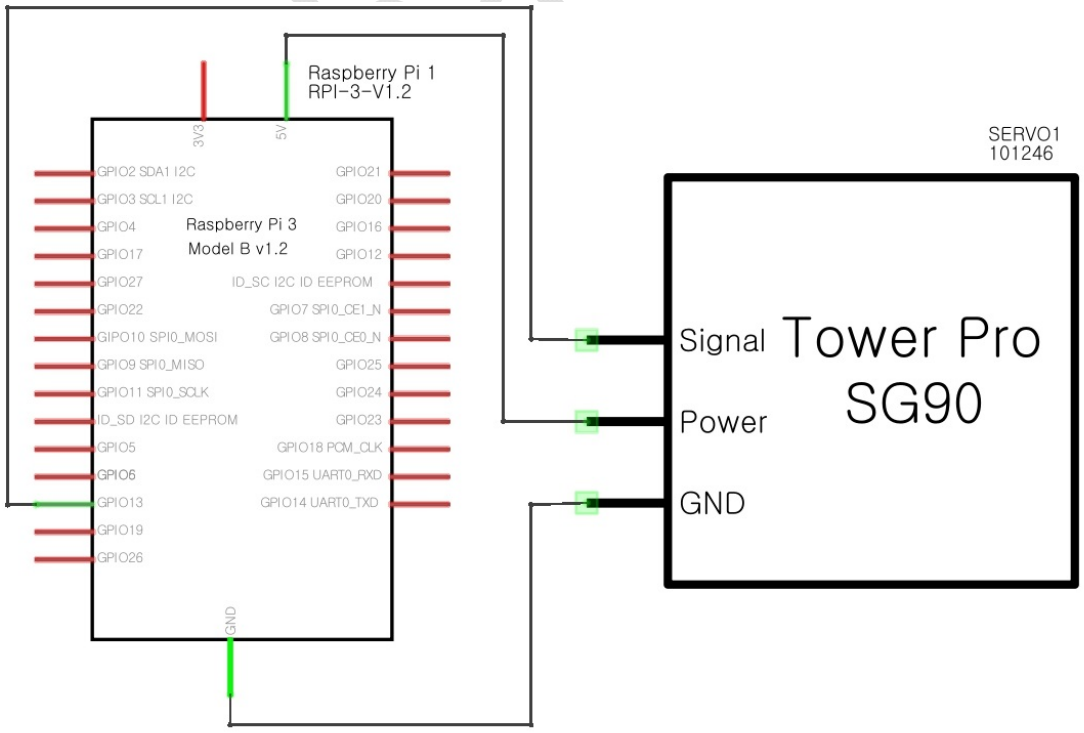
5.4.2 회로 구성하기

아래와 같이 회로를 구성합니다.

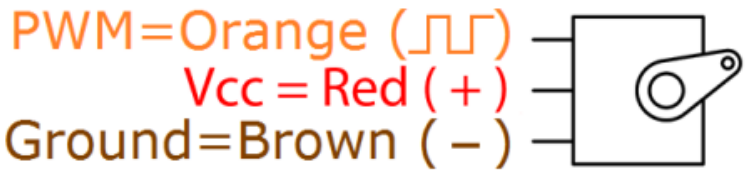




라즈베리파이에는 2개의 하드웨어 PWM 핀이 있습니다. PWM0은 GPIO18이고, PWM1은 GPIO13입니다. 라즈비안에서는 소프트웨어 PWM을 사용하면 거의 모든 GPIO핀을 PWM으로 사용할 수 있으나, 안드로이드 씽스에서는 아직 지원하고 있지 않습니다. 그렇기 때문에 많은 수의 모터를 제어하고자 할 때는 별도의 PWM모듈을 사용해야 합니다.



아래는 SG90의 선입니다. 일종의 RC용 표준으로 RC용 서보모터나 DC모터 또는 BLDC모터 변속기(ESC)는 모두 이런 형태의 배치를 사용합니다. 출력이 높은 서보 모터를 사용할 경우 Vcc는 외부 전원에 연결하여야 합니다.



5.4.3 코드 작성하기

그래들 설정

dependencies에 다음을 추가합니다.

```
dependencies {
    ...

    compile 'com.google.android.things.contrib:driver-pwmservo:0.3'
    provided 'com.google.android.things:androidthings:0.5.1-devpreview'
}
```

```
21
22      dependencies {
23          compile fileTree(dir: 'libs', include: ['*.jar'])
24          androidTestCompile('com.android.support.test.espresso:espresso-core:2.2.2', {
25              exclude group: 'com.android.support', module: 'support-annotations'
26          })
27          compile 'com.android.support:appcompat-v7:26.+'
28          compile 'com.android.support.constraint:constraint-layout:1.0.2'
29          testCompile 'junit:junit:4.12'
30
31          compile 'com.google.android.things.contrib:driver-pwmservo:0.3'
32          provided 'com.google.android.things:androidthings:0.5.1-devpreview'
33      }
34
```

매니페스트 설정

아래와 같이 <application .... >이 끝나는 부분에 아래 코드를 입력합니다.

```
<application>
    <uses-library android:name="com.google.android.things"/>

    ...
</application>
```

```
4
5       <application
6           android:allowBackup="true"
7           android:icon="@mipmap/ic_launcher"
8           android:label="@string/app_name"
9           android:roundIcon="@mipmap/ic_launcher_round"
10          android:supportsRtl="true"
11          android:theme="@style/AppTheme" >
12
13          <uses-library android:name="com.google.android.things"/>
14
15          <activity android:name=".MainActivity" >
```

코드는 LED 켜고 끄기와 마찬가지로 쓰레드를 사용합니다.

서보 모터 제어를 위해 Servo 클래스 객체와 이를 주기적으로 제어하기 위해 쓰레드, 러너블 클래스 객체를 선언해줍니다. While문 내부는 후에 채우도록 하겠습니다.

```
private Servo mServo;
Thread mThread;
Runnable mRunnable = new Runnable() {
    public void run() {
        while (true) {

        }
    }
};
```

```
10  public class MainActivity extends Activity {
11      private Servo mServo;
12      Thread mThread;
13      Runnable mRunnable = new Runnable() {
14          public void run() {
15
16          }
17      };
18
```

앞서 정리한 서보 모터 제어 방법을 다시 정리해보면, PWM의 주파수는 50Hz, 주기가 20ms에 PWM의 폭이 1ms 일 때 -90도, 2ms 일 때 90도, 1.5ms 일 때 0도 입니다. 이것을 서보 클래스를 초기화할 때 펄스폭은 1~2ms, 각도 폭은 -90~90도로 설정해줍니다. 문제는 펄스폭 부분이 정확히 맞지가 않습니다. 심지어 같은 SG90 서보 모터인데도 서로 조금씩 다릅니다. 아래 코드는 필자가 값을 조금씩 바꿔보면서 맞춘 것으로 펄스폭이 원래 1~2ms이어야 했던 것을 0.75~2.6ms로 조정했습니다. 여러분들도 실제 하시려면 값을 조금 조정하셔야 할 것입니다.

```
try {
    mServo = new Servo("PWM1");
    mServo.setPulseDurationRange(0.75,2.6);
    mServo.setAngleRange(-90, 90);
    mServo.setEnabled(true);
} catch (IOException e) {}

mThread = new Thread(mRunnable);
mThread.start();
```

```
35      @Override
36      protected void onCreate(Bundle savedInstanceState) {
37          super.onCreate(savedInstanceState);
38          setContentView(R.layout.activity_main);
39
40          try {
41              mServo = new Servo("PWM0");
42              mServo.setPulseDurationRange(0.75,2.6);
43              mServo.setAngleRange(-90, 90);
44              mServo.setEnabled(true);
45          } catch (IOException e) {}
46
47          mThread = new Thread(mRunnable);
48          mThread.start();
49      }
50  }
```

처음 멤버 객체/변수 작성 부분에서 비워놓았던 While 문 내부를 여기서 채우도록 하겠습니다. Servo 클래스의 setAngle 함수를 이용해서 서보 모터의 각도값을 입력해줍니다. 첫 루프에서 -90도로 출발한 다음 2초를 대기하고 다시 90도로 출발한 다음 2초 대기하기를 반복하는 코드입니다.

```
while (true) {
    try {
        mServo.setAngle(-90);
    } catch (IOException e) {}

    try {
        Thread.sleep(2000);
    } catch (InterruptedException e) {}

    try {
        mServo.setAngle(90);
    } catch (IOException e) {}
```

```
        try {
            Thread.sleep(2000);
        } catch (InterruptedException e) {}
    }
```

```
10  public class MainActivity extends Activity {
11      private Servo mServo;
12      Thread mThread;
13      Runnable mRunnable = new Runnable() {
14          public void run() {
15              while (true) {
16                  try {
17                      mServo.setAngle(-90);
18                  } catch (IOException e) {}
19
20                  try {
21                      Thread.sleep(2000);
22                  } catch (InterruptedException e) {}
23
24                  try {
25                      mServo.setAngle(90);
26                  } catch (IOException e) {}
27
28                  try {
29                      Thread.sleep(2000);
30                  } catch (InterruptedException e) {}
31              }
32          }
33      };
```

아래는 MainActivity.java 파일의 전체 내용입니다. 맨 위의 패키지 부분은 혼란의 여유가 있어서 포함시키지 않았습니다.

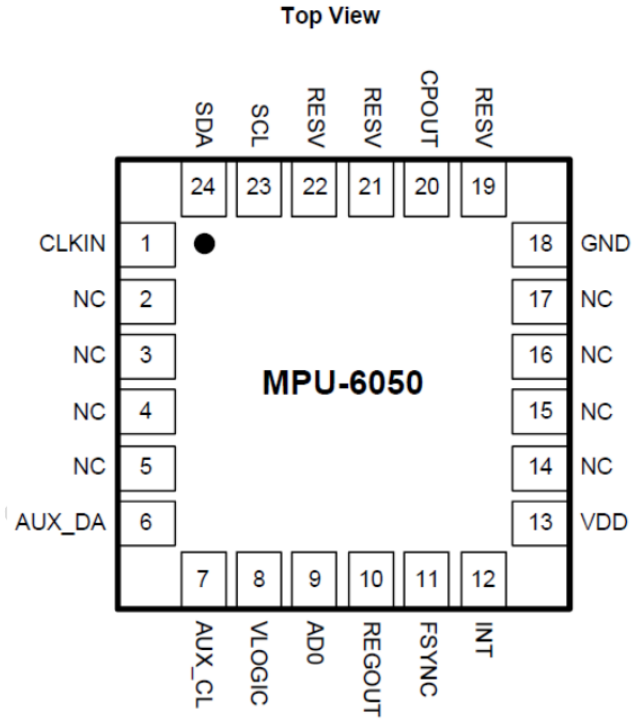
```
import android.app.Activity;
import android.os.Bundle;

import com.google.android.things.contrib.driver.pwmservo.Servo;

import java.io.IOException;

public class MainActivity extends Activity {
    private Servo mServo;
    Thread mThread;
    Runnable mRunnable = new Runnable() {
        public void run() {
            while (true) {
                try {
                    mServo.setAngle(-90);
```

```
                } catch (IOException e) {}

                try {
                    Thread.sleep(2000);
                } catch (InterruptedException e) {}

                try {
                    mServo.setAngle(90);
                } catch (IOException e) {}

                try {
                    Thread.sleep(2000);
                } catch (InterruptedException e) {}
            }
        }
    };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        try {
            mServo = new Servo("PWM0");
            mServo.setPulseDurationRange(0.75,2.6);
            mServo.setAngleRange(-90, 90);
            mServo.setEnabled(true);
        } catch (IOException e) {}

        mThread = new Thread(mRunnable);
        mThread.start();
    }
}
```

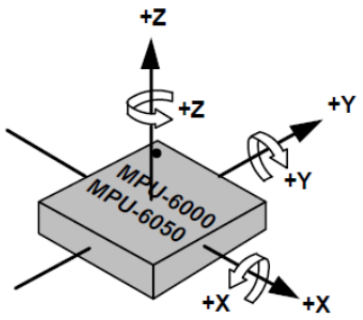
5.5 I2C 통신으로 6축 센서 입력 받기

모바일 시스템에서 장치의 자세 및 움직이는 상태, 충격, 진동 등을 감지하기 위해 6축 센서라는 것을 사용합니다. 일반적으로 물체의 운동을 설명할 때, x, y, z 방향으로의 직선 운동과, x, y, z 축을 기준으로 한 회전 운동을 이야기 합니다. 물리학적으로 외부의 도움 없이 움직이는 물체 내부에서 이 값들의 측정은 직선 운동에서의 가속도와, 회전 운동에서의 각속도에 제한됩니다. 3방향으로 직선 운동의 가속도를 측정하는 센서를 3축 가속도 센서, 3방향으로 회전 운동의 각속도를 측정하는 센서를 3축 자이로 센서라고 합니다.

스마트 폰에도 이런 센서가 있어서 폰을 가로로 보고 있는지, 세로로 보고 있는지를 판단해서 화면을 회전시켜주기도 하고, 스마트 폰의 기울기를 이용해서 게임을 하기도 합니다. 6축에 전자나침반과 온도의 영향까지 계산하기 위해 10축 센서를 사용하기도 합니다. 이런 센서들은 자동차나 선박, 항공, 우주, 군사 등 다양한 분야에서 사용되며 가격대도 천차만별입니다. 아래는 저렴한 가격으로 많이 사용되고 있는 6축 센서 MPU 6050입니다.



QFN Package
24-pin, 4mm x 4mm x 0.9mm

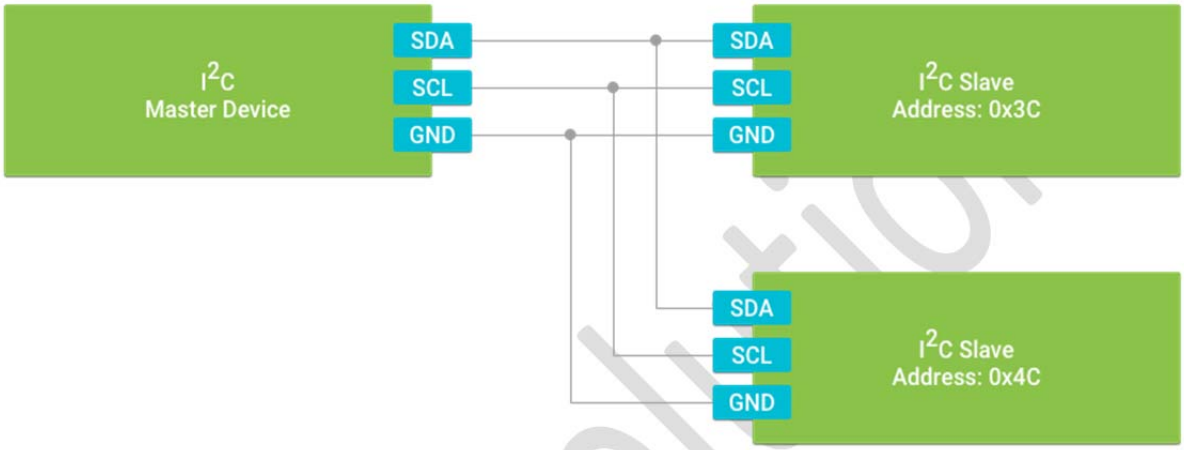


Orientation of Axes of Sensitivity and
Polarity of Rotation

오른쪽 그림과 같이 3방향으로 직선 가속도와 회전 각속도를 측정합니다. 추가적으로 온도 측정도 할 수 있는데, 온도 자체를 측정하는 목적보다는, 온도에 따라서 센서의 출력이 영향을 받기 때문에 이를 잡아주기 위한 목적으로 사용됩니다.

5.5.1 I2C 통신 이해하기

I2C(Inter-Integrated Circuit) 통신은 아두이노 등 SPI와 함께 임베디드에서 주변 장치와 작은 용량의 데이터를 주고 받을 때, 이용되는 통신 방법입니다. 지금처럼 가속도/각속도 센서나 LCD모니터, 모터 드라이버 등 중급 이상 수준의 모듈을 사용할 때 많이 이용됩니다. 구조는 아래와 같습니다.



I2C 통신은 그림에서 보는 것처럼 하나의 버스에 여러 개의 I2C 장치를 연결해서 사용할 수 있습니다. I2C 통신은 3개의 선을 필요로 하는데, 데이터를 주고 받는 SDA(Shared Data Line), 통신 속도를 결정하는 SCL(Shared Clock Signal) 그리고 전압의 기준점이 되는 GND입니다. 여기에 추가로 장치에 별도의 전원이 필요할 경우 그에 맞는 전원을 공급해줍니다.

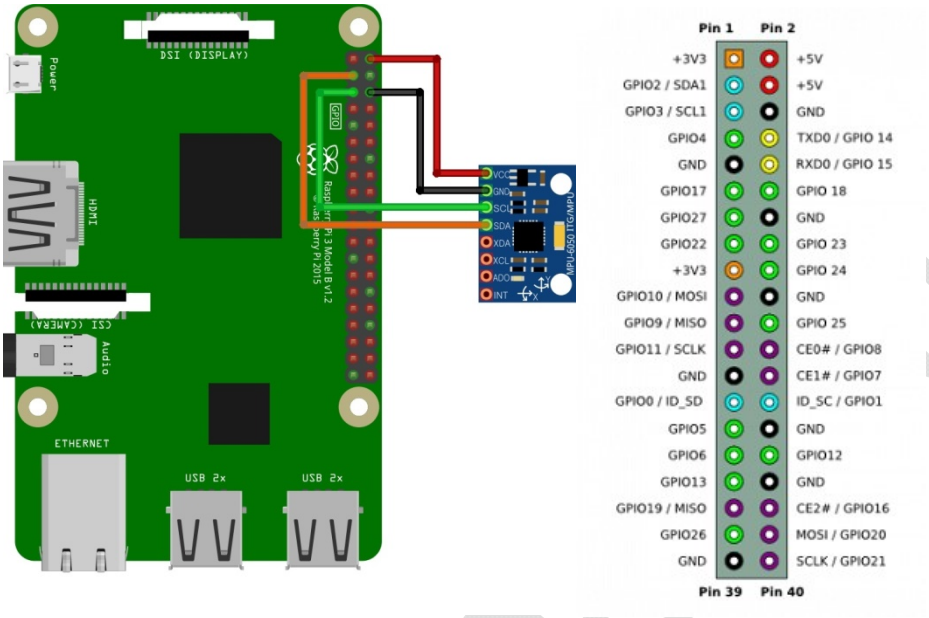
버스를 공유하는 구조이기 때문에, 각 장치를 사용할 때는 장치마다 가지고 있는 고유의 주소를 이용합니다. 예를 들면 그림에서 보는 것처럼 0x3C, 0x4C 같은 형태가 될 수 있으며, 첫째자리는 8bit, 둘째자리는 7bit이기 때문에 이론적으로는 32768개의 장치를 사용할 수 있으나, 장치 간의 충돌 문제로 물리적으로나 시스템 상으로나 2개 이상도 구성이 쉬운 편은 아닙니다.

이러한 장치 주소와 함께 레지스터 주소가 있습니다. 레지스터란 일종의 메모리로 I2C 통신에서는 16bit로 되어있는 레지스터 주소를 입력하면, 해당 레지스터의 값을 읽거나 쓸 수 있습니다.

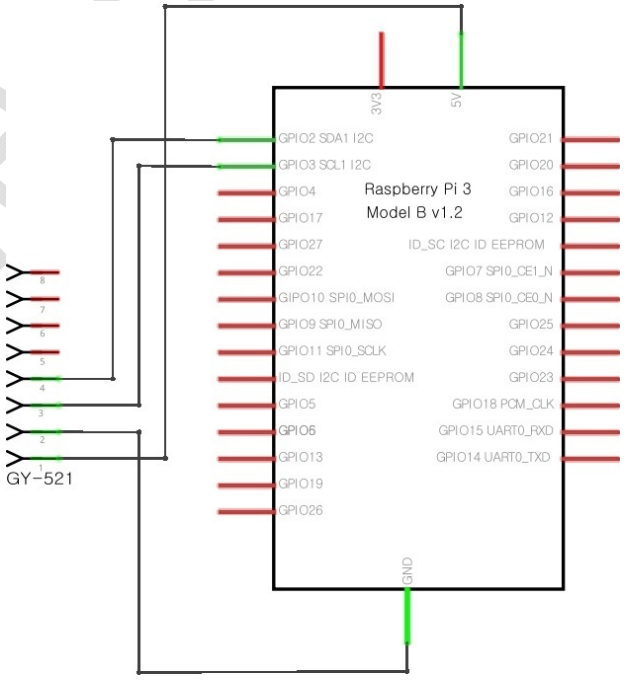
일반적으로 I2C 통신을 이용하려면, I2C 통신과 함께, 장치 자체에 대해서도 잘 알아야하기 때문에 코딩을 할 때 직접 손을 대기 보다는, 이미 해당 장치에 맞게 완성된 라이브러리를 이용하는 경우가 많습니다. 하지만 안드로이드 씽스가 아직 초창기인 관계로 아직은 지원되는 라이브러리가 많지는 않으므로 이번 장에서는 처음부터 하나씩 해보도록 하겠습니다.

5.5.2 회로 구성하기

아래와 같이 회로를 구성합니다.



MPU6050은 센서칩의 이름이고, 일반적으로 MPU6050을 주변의 보호회로 등과 PCB 기판 위에 올린 GY-521이라는 모듈을 사용하며, 통상적으로 GY-521보다는 MPU6050으로 부릅니다. 라즈베리파이의 GPIO2/SDA1은 MPU6050의 SDA에 연결되고, GPIO3/SCL1은 MPU6050의 SCL에 연결됩니다.



5.5.3 코드 작성하기

MPU6050의 데이터 시트와 레지스터 맵은 아래 링크에서 볼 수 있습니다.

https://store.invensense.com/datasheets/invensense/MPU-6050_DataSheet_V3%204.pdf

https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Register-Map1.pdf

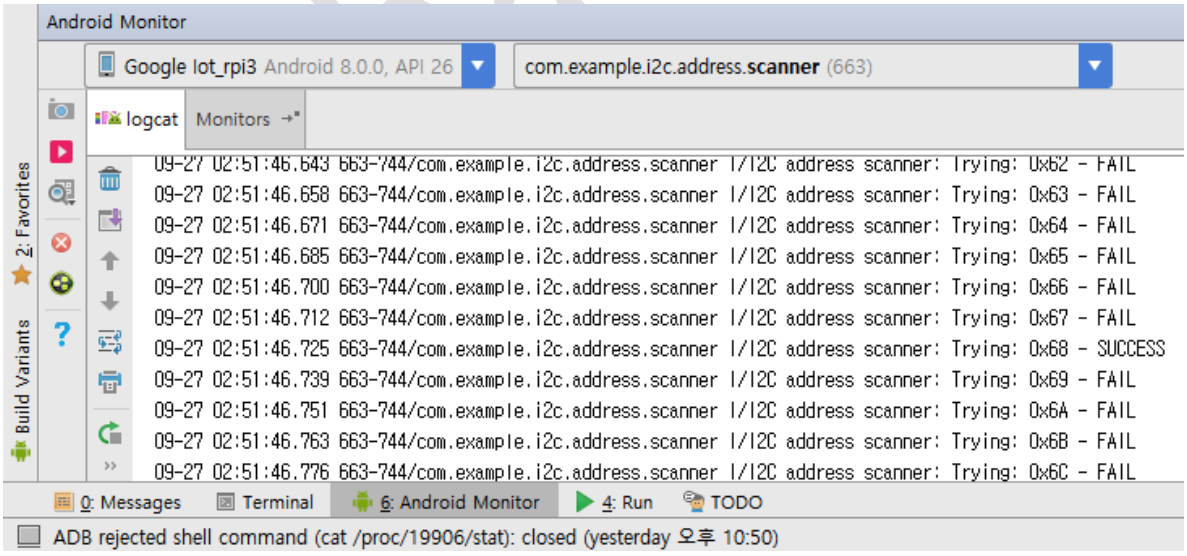
MPU6050의 장치 주소는 0x68입니다만, 데이터 시트에서 찾기도 쉽지 않고, 주소를 알더라도 장치의 정상 작동 여부를 확인할 필요가 있기 때문에 아래 링크의 i2c-address-scanner를 사용하면 연결되 i2c 장치와 그 주소를 확인할 수 있습니다.

https://github.com/dennisg/i2c-address-scanner

우측의 녹색 Clone or download 버튼을 누른 후 Download Zip을 누르시거나 아래 링크로 다운로드를 받으시면 됩니다.

https://github.com/dennisg/i2c-address-scanner/archive/master.zip

다운로드 받으시고 압축을 해제하신 후에 안드로이드 스튜디오에서 [Menu]-[File]-[Open]을 누릅니다. 압축을 해제한 폴더로 찾아가서 OK를 누르면 프로젝트가 열립니다. MPU6050이 연결된 상태에서 업로드를 하시면 Android Monitor에서 다음과 같은 결과를 확인할 수 있습니다.



여기서는 다루지 않으나 비교적 단순하면서 잘 작성된 코드이므로 여유가 되실 때 살펴보실 것을 추천드립니다. 좋은 코드를 많이 보면 코딩 실력을 높이는데 많은 도움이 됩니다.

그래들 설정

아래 쪽 dependencies에 다음 내용을 아래 그림과 같이 추가해줍니다. GPIO는 기존의 안드로이드에 포함되어 있는 내용이 아니기 때문에 Things Support Library를 추가해줘야 합니다.

```
dependencies {
    ...
    provided 'com.google.android.things:androidthings:0.5.1-devpreview'
}
```

```
21
22      dependencies {
23          compile fileTree(dir: 'libs', include: ['*.jar'])
24          androidTestCompile('com.android.support.test.espresso:espresso-core:2.2.2', {
25              exclude group: 'com.android.support', module: 'support-annotations'
26          })
27          compile 'com.android.support:appcompat-v7:26.+'
28          compile 'com.android.support.constraint:constraint-layout:1.0.2'
29          testCompile 'junit:junit:4.12'
30
31          provided 'com.google.android.things:androidthings:0.5.1-devpreview'
32      }
33
```

매니페스트 설정

아래와 같이 <application .... >이 끝나는 부분에 아래 코드를 입력합니다.

```
<application>
    <uses-library android:name="com.google.android.things"/>

    ...
</application>
```

```
4
5       <application
6           android:allowBackup="true"
7           android:icon="@mipmap/ic_launcher"
8           android:label="@string/app_name"
9           android:roundIcon="@mipmap/ic_launcher_round"
10          android:supportsRtl="true"
11          android:theme="@style/AppTheme" >
12
13          <uses-library android:name="com.google.android.things"/>
14
15          <activity android:name=".MainActivity" >
```

주기적으로 MPU6050의 값을 읽어오기 위해 쓰레드 클래스 객체를 선언해줍니다. While문 내부는 후에 채우도록 하겠습니다.

```
private Servo mServo;
I2cDevice mDevice
Thread mThread = new Thread() {
    public void run() {
        while (true) {

        }
    }
};
```

```
11  public class MainActivity extends Activity {
12      I2cDevice mDevice;
13      Thread mThread = new Thread() {
14          public void run() {
15              while (true) {
16
17              }
18          }
19      };
```

초기화를 하려면 우선 PeripheralManagerService 객체를 통해 I2cDevice 객체를 얻어옵니다. 이때 버스 이름은 "I2C1"이며, 장치 주소는 앞에서 찾은대로 0x68입니다. I2cDevice 객체를 얻어오면 본격적으로 장치와 통신할 수 있는데 이 시점에서 장치와 내부 레지스터에 대한 이해가 필요합니다. 다행히 MPU6050의 초기화에는 0x6B 레지스터만 설정해주면 됩니다.

4.28 Register 107 – Power Management 1
PWR_MGMT_1

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
6B	107	DEVICE _RESET	SLEEP	CYCLE	-	TEMP_DIS	CLKSEL[2:0]		

처음 전원이 들어오면 MPU6050이 Sleep모드로 되어있습니다. 위 0x6B 레지스터의 6번 bit가 1로 되어있는데 이것을 0으로 바꿔주면 됩니다. 6번 비트만 바꾸려면 다소 번거로운 부분이 있지만, 나머지 비트들도 전부 0이므로, 레지스터 자체를 0x00으로 바꿔주면 됩니다. OnCreate 함수는 다음과 같습니다.

```
PeripheralManagerService pioService = new PeripheralManagerService();
try {
    mDevice = pioService.openI2cDevice("I2C1", 0x68);
    mDevice.writeRegByte(0x6B, (byte) 0x00);
```

```
} catch (IOException e) {
      e.printStackTrace();
}

mThread.start();
```

```
50      @Override
51      protected void onCreate(Bundle savedInstanceState) {
52          super.onCreate(savedInstanceState);
53          setContentView(R.layout.activity_main);
54
55          PeripheralManagerService pioService = new PeripheralManagerService();
56          try {
57              mDevice = pioService.openI2cDevice("I2C1", 0x68);
58              mDevice.writeRegByte(0x6B, (byte) 0x00);
59          } catch (IOException e) {
60              e.printStackTrace();
61          }
62
63          mThread.start();
64      }
```

이 장에서는 1초 간격으로 MPU6050의 값을 반복적으로 받아올 것입니다. 먼저 MPU6050에서 가속도와 온도, 각속도가 저장되는 레지스터의 주소는 다음과 같습니다.

Addr (Hex)	Addr (Dec.)	Register Name	Serial I/F	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
3B	59	ACCEL_XOUT_H	R	ACCEL_XOUT[15:8]							
3C	60	ACCEL_XOUT_L	R	ACCEL_XOUT[7:0]							
3D	61	ACCEL_YOUT_H	R	ACCEL_YOUT[15:8]							
3E	62	ACCEL_YOUT_L	R	ACCEL_YOUT[7:0]							
3F	63	ACCEL_ZOUT_H	R	ACCEL_ZOUT[15:8]							
40	64	ACCEL_ZOUT_L	R	ACCEL_ZOUT[7:0]							
41	65	TEMP_OUT_H	R	TEMP_OUT[15:8]							
42	66	TEMP_OUT_L	R	TEMP_OUT[7:0]							
43	67	GYRO_XOUT_H	R	GYRO_XOUT[15:8]							
44	68	GYRO_XOUT_L	R	GYRO_XOUT[7:0]							
45	69	GYRO_YOUT_H	R	GYRO_YOUT[15:8]							
46	70	GYRO_YOUT_L	R	GYRO_YOUT[7:0]							
47	71	GYRO_ZOUT_H	R	GYRO_ZOUT[15:8]							
48	72	GYRO_ZOUT_L	R	GYRO_ZOUT[7:0]							

먼저 우리에게 필요한 데이터는 레지스터 주소 3B~48까지 저장되어있는 것을 알 수 있습니다. 이것을 I2cDevice 클래스의 readRegBuffer 함수를 사용하면 한번에 가져올 수 있습니다. 그 다음으로는 MPU6050은 16비트 분해능 센서이기 때문에 하나의 값을 2바이트로 나눠서 저장하고 레지스터도 2개가 사용됩니다. 위 표를 보면 가속도 센서의 X값이 3B와 3C에 나뉘어서 담겨 있습니다. 이것을 다시 합쳐주는 함수가 필요하게 됩니다. 다시 합쳐주게 되면 +- 부호를 제외하고 최

고값이 2^15 = 32768인 값이 나옵니다. MPU6050은 다양한 측정범위를 지원하는데, 기본값은 가속도 ±2g, 각속도는 ±250º/s까지 측정하도록 설정되어 있습니다. 가속도의 g값은 중력가속도로, MPU6050의 X축이 지면으로부터 수직일 때 가속도는 -1g가 되게 됩니다. 마찬가지로 각속도의 경우 1초에 반바퀴를 회전하고 있다면, 각속도가 180º/s가 되게 됩니다.

가속도의 경우 측정 최고 값이 출력 최고 값에 대응되므로 데이터 값을 32768로 나눠준 후 다시 2로 나눠주면 되며, 각속도의 경우 측정 최대값이 250을 곱해준 후 32768로 나눠주면 됩니다. 이런 것들을 계산하고 출력하는 코드는 다음과 같습니다. 온도 값의 경우 데이터 시트에서 제공하는 공식을 사용하였습니다.

```
Thread mThread = new Thread() {
    public void run() {
        byte[] buffer = new byte[14];

        while (true) {
            try {
                mDevice.readRegBuffer(0x3B, buffer, 14);
            } catch (IOException e) {
                e.printStackTrace();
            }

            double acX = (double) appendByte(buffer[0], buffer[1]) / 32768. / 2.;
            double acY = (double) appendByte(buffer[2], buffer[3]) / 32768. / 2.;
            double acZ = (double) appendByte(buffer[4], buffer[5]) / 32768. / 2.;
            double Tmp = (double) appendByte(buffer[6], buffer[7]) / 340. + 36.53;
            double gyX = (double) appendByte(buffer[8], buffer[9]) * 250. / 32768.;
            double gyY = (double) appendByte(buffer[10], buffer[11]) * 250. / 32768.;
            double gyZ = (double) appendByte(buffer[12], buffer[13]) * 250. / 32768.;

            Log.i("Acel", String.format("%f \t %f \t %f", acX, acY, acZ));
            Log.i("Temp", String.format("%f", Tmp));
            Log.i("Gyro", String.format("%f \t %f \t %f", gyX, gyY, gyZ));

            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
};
```

```
public short appendByte(byte a, byte b) {
    return (short) (((a << 8) & 0xff00) | (b & 0x00ff));
}
```

```
13      I2cDevice mDevice;
14      Thread mThread = new Thread() {
15          public void run() {
16              byte[] buffer = new byte[14];
17              while (true) {
18                  try {
19                      mDevice.readRegBuffer(0x3B, buffer, 14);
20                  } catch (IOException e) {
21                      e.printStackTrace();
22                  }
23
24                  double acX = (double) appendByte(buffer[0], buffer[1]) / 32768. / 2.;
25                  double acY = (double) appendByte(buffer[2], buffer[3]) / 32768. / 2.;
26                  double acZ = (double) appendByte(buffer[4], buffer[5]) / 32768. / 2.;
27                  double Tmp = (double) appendByte(buffer[6], buffer[7]) / 340. + 36.53;
28                  double gyX = (double) appendByte(buffer[8], buffer[9]) * 250. / 32768.;
29                  double gyY = (double) appendByte(buffer[10], buffer[11]) * 250. / 32768.;
30                  double gyZ = (double) appendByte(buffer[12], buffer[13]) * 250. / 32768.;
31
32                  Log.i("Acel", String.format("%f \t %f \t %f", acX, acY, acZ));
33                  Log.i("Temp", String.format("%f", Tmp));
34                  Log.i("Gyro", String.format("%f \t %f \t %f", gyX, gyY, gyZ));
35
36                  try {
37                      Thread.sleep(1000);
38                  } catch (InterruptedException e) {
39                      e.printStackTrace();
40                  }
41              }
42          }
43      };
44
45      public short appendByte(byte a, byte b) {
46          return (short) (((a << 8) & 0xff00) | (b & 0x00ff));
47      }
```

전체 코드는 다음과 같습니다.

```
import android.app.Activity;
import android.os.Bundle;
import android.util.Log;
```

```
import com.google.android.things.pio.I2cDevice;
import com.google.android.things.pio.PeripheralManagerService;

import java.io.IOException;

public class MainActivity extends Activity {
    I2cDevice mDevice;
    Thread mThread = new Thread() {
        public void run() {
            byte[] buffer = new byte[14];
            while (true) {
                try {
                    mDevice.readRegBuffer(0x3B, buffer, 14);
                } catch (IOException e) {
                    e.printStackTrace();
                }

                double acX = (double) appendByte(buffer[0], buffer[1]) / 32768. / 2.;
                double acY = (double) appendByte(buffer[2], buffer[3]) / 32768. / 2.;
                double acZ = (double) appendByte(buffer[4], buffer[5]) / 32768. / 2.;
                double Tmp = (double) appendByte(buffer[6], buffer[7]) / 340. + 36.53;
                double gyX = (double) appendByte(buffer[8], buffer[9]) * 250. / 32768.;
                double gyY = (double) appendByte(buffer[10], buffer[11]) * 250. / 32768.;
                double gyZ = (double) appendByte(buffer[12], buffer[13]) * 250. / 32768.;

                Log.i("Acel", String.format("%f \t %f \t %f", acX, acY, acZ));
                Log.i("Temp", String.format("%f", Tmp));
                Log.i("Gyro", String.format("%f \t %f \t %f", gyX, gyY, gyZ));

                try {
                    Thread.sleep(1000);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }
    };

    public short appendByte(byte a, byte b) {
        return (short) (((a << 8) & 0xff00) | (b & 0x00ff));
    }

    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        PeripheralManagerService pioService = new PeripheralManagerService();
        try {
            mDevice = pioService.openI2cDevice("I2C1", 0x68);
            mDevice.writeRegByte(0x6B, (byte) 0x00);
        } catch (IOException e) {
            e.printStackTrace();
        }

        mThread.start();
    }
}
```