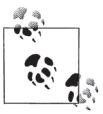
CHAPTER 1

Arduino Cookbook, Second Edition

시작하기

1.0 소개

아두이노 환경은 소프트웨어나 전자 공학과 관련된 경험이 없는 초보자도 쉽게 사용할 수 있도록 설계되었다. 아두이노를 사용하면 빛, 소리, 접촉, 움직임 등에 반응하거나 이러한 것들을 제어할 수 있는 도구를 만들 수 있다. 아두이노를 사용하여 만들 수 있는 것의 예로는 악기, 로봇, 빛 조형, 게임, 인터랙티브 가구 등이 있으며, 심지어는 인터랙티브 의상도 만들 수 있다.



처음 시작하는 경우가 아니라면 흥미를 끄는 레시피부터 바로 시작해도 괜찮다.

아두이노는 전 세계적으로 수많은 교육 프로그램에서 사용되고 있으며, 특히 기술에 대한 깊은 이해 없이도 프로토타입을 쉽게 만들고자 하는 디자이너와 예술가들이 많이 사용하고 있다. 기술이 없는 사람도 사용할 수 있도록 설계되었기 때문에 아두이노 소프트웨어에는 아두이노 보드의 다양한 기능을 활용하는 방법을 보여 주는 풍부한 예제 코드가 담겨 있다.

아두이노의 기본 하드웨어는 사용하기 쉬움에도 불구하고 임베디드 장치를 만들 수 있을 정도로 높은 수준의 정교함을 갖추고 있다. 마이크로컨트롤러를 다루는 사람들은 이미 아이디어를 빠르게 구현할 수 있는 아두이노의 뛰어난 개발 환경과 그 기능

에 매료되어 있다.

아두이노는 하드웨어로 잘 알려져 있지만 이 하드웨어를 프로그래밍하기 위한 소프트웨어도 필요하다. 하드웨어와 소프트웨어 모두 "아두이노(Arduino)"라고 부른다. 이러한 하드웨어와 소프트웨어를 함께 사용하여 실제 세계를 감지하고 제어하는 프로젝트를 만들 수 있다. 이 소프트웨어는 무료로 제공되는 오픈 소스 크로스-플랫폼이다. 약간의 비용을 지불하고 보드를 구매하거나 하드웨어 설계가 오픈 소스로 공개되어 있으므로 자신이 직접 보드를 제작할 수도 있다. 그리고 아두이노 포럼과 아두이노 플레이그라운드(Arduino Playground)라는 위키를 통해 활발한 활동이 이루어지고 있는 아두이노 커뮤니티에 참여할 수 있다. 이러한 포럼과 위키에서는 프로젝트 개발 예제와 여러 가지 문제에 대한 해결책을 제공하고 있으므로 프로젝트를 개발할 때 많은 아이디어와 도움을 얻을 수 있을 것이다. 이 장의 레시피에서는 개발 환경을 설정하는 방법과 예제 스케치를 컴파일하고 실행하는 방법에 대해 설명한다.

아두이노 커뮤니티에서는 아두이노 기능을 제어하는 컴퓨터 명령이 포함된 소스 코드를 보통 스케치(sketch)라고 부른다. 이 책에서는 스케치라는 용어를 사용하여 아두이노 프로그램 코드를 지칭한다.

이 장의 레시피에서는 아두이노와 함께 제공되는 Blink 스케치를 예제로 사용한다. 하지만 이 장의 마지막 레시피에서는 단순히 보드에 내장된 조명을 깜박이기만 하는 것이 아니라 사운드를 추가하고 추가 하드웨어를 통해 입력을 수집하는 방법까지도 설명한다. 2장에서는 아두이노의 스케치를 구조화하는 방법과 함께 기본적인 프로그래밍 방법을 살펴본다.

아두이노와 관련된 기본적인 사항을 이미 알고 있다면 원하는 주제를 찾아서 살펴보는 것도 좋을 것이다. 하지만 아두이노를 처음으로 접하는 사용자라면 인내심을 갖고 전반부의 레시피를 따라하다 보면 나중에 좋은 결과를 얻게 될 것이다.

아두이노 소프트웨어

스케치(sketch)라고 하는 소프트웨어 프로그램은 컴퓨터에 설치된 아두이노 통합 개발 환경(IDE)을 사용하여 만들어진다. 이 IDE에서는 코드를 작성 및 편집할 수 있으며, 아두이노 하드웨어에서 이해할 수 있는 명령어로 코드를 변환할 수 있다. 또한

변환된 명령어를 아두이노 보드로 전송할 수도 있다(업로드(upload) 프로세스).

아두이노 하드웨어

아두이노 보드에서는 사용자가 작성한 코드가 실제로 실행된다. 이 보드는 전기 신호를 제어하고 그에 따라 반응만 할 수 있으므로, 실질적으로 어떤 작업을 수행하려면 특정 부품을 보드에 연결해야 한다. 예를 들어, 실제 사물의 특성을 전기 신호로 변환하는 센서를 보드에 연결해서 사물의 특성을 감지하거나, 보드의 전기 신호를 받아서 특정 동작으로 변환해 주는 액츄에이터를 연결해서 원하는 작업을 수행할 수 있다. 센서의 예로는 스위치, 가속도계, 초음파 거리 센서 등이 있다. 액츄에이터의 예로는 조명과 LED, 스피커, 모터, 디스플레이 등이 있다.

아두이노 소프트웨어와 함께 사용할 수 있는 여러 가지 공식 보드뿐만 아니라 커뮤니티에서 제작한 아두이노 호환 보드도 많이 있다. 전원을 공급하고 소프트웨어를 보드에 업로드하는 데 사용되는 USB 커넥터가 장착된 보드가 가장 많이 사용되고 있으며, 그림 1-1에서는 대부분의 사람들이 시작할 때 사용하는 기본 보드인 Arduino Uno를 보여 준다.

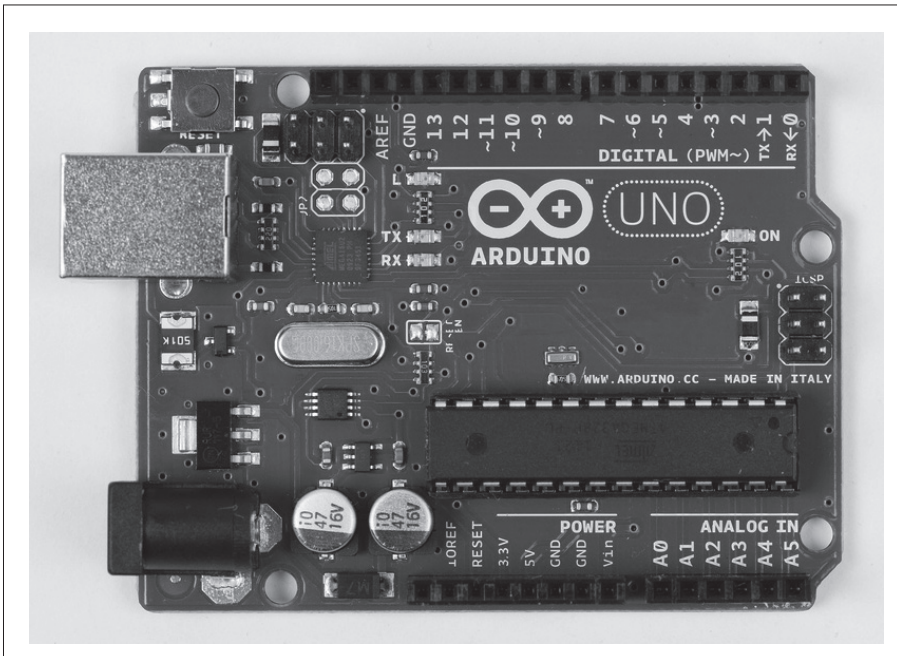


그림 1-1 기본 보드: Arduino Uno. 사진 제공: todo.to.it.

Arduino Uno에는 모든 USB 통신을 처리하는 보조 마이크로컨트롤러가 있으며, 이 작은 칩(ATmega8U2)은 보드의 USB 소켓 근처에 있다. 이 칩은 보드를 다양한 USB 장치로 표시하고 싶을 때 별도로 프로그래밍할 수 있다(레시피 18.14). Arduino Leonardo 보드에서는 ATmega8U2 및 ATmega328 컨트롤러를 빼고 USB 프로토콜을 소프트웨어로 구현한 하나의 ATmega32u4 칩을 사용한다. 아두이노와 호환되는 PJRC의 Teensy 및 Teensy+ 보드(http://www.pjrc.com/teensy/)도 USB 장치를 에뮬레이트할 수 있다. 이전 버전의 보드와 대부분의 아두이노 호환 보드에서는 컴퓨터의 시리얼 포트에 연결되는 하드웨어 USB 솔루션을 갖추고 있는 FTDI 회사의 칩을 사용하고 있다.

Arduino Mini 및 Pro Mini는 소형 보드이며, Arduino Mega는 좀 더 크면서 연결 옵션이 많고 프로세서 성능이 높다. 그리고 특정 분야를 위한 맞춤형 보드도 있다. 맞춤형 보드로는 의상 분야를 위한 LilyPad, 무선 프로젝트를 위한 Fio, 임베디드 분야(주로 배터리로 작동하는 독립형 프로젝트)를 위한 Arduino Pro가 있다.

최근에 추가된 제품인 Arduino ADK에는 USB 호스트 소켓이 장착되어 있으며, Android Open Accessory Development Kit와 호환된다. 공식적으로 승인을 받은 이 개발 키트는 안드로이드 장치에 하드웨어를 연결하는 기능을 제공한다. Leonardo 보드에는 보드 자체를 다양한 HID 장치로 표현할 수 있는 컨트롤러 칩(ATmega32u4)이 장착되어 있다. Ethernet 보드에는 이더넷 연결이 포함되어 있다. 그리고 Power Over Ethernet 옵션이 제공되므로 보드 연결과 전원 공급이 단일 배선만으로도 가능하다.

이외에도 아래 목록에 있는 보드를 포함한 여러 가지 아두이노 호환 보드가 있다.

- Arduino Nano, USB 기능을 갖춘 소형 보드, Gravitech(http://store.gravitech.us/arna30wiatn.html)
- Bare Bones Board, USB 기능 유무를 선택할 수 있는 경제형 보드, Modern Device(http://www.moderndevice.com/products/bbb-kit)
- Boarduino, 경제적인 브레드보드 호환 보드, Adafruit Industries(http://www.adafruit.com/)
- Seeeduino, 다양한 종류의 표준 USB 보드, Seeed Studio Bazaar(http://www.seeedstudio.com/)

- Teensy 및 Teensy++, 작지만 다양한 기능을 제공하는 보드, PJRC(http://www.pjrc.com/teensy/)

http://www.freeduino.org/에서 아두이노 호환 보드 목록을 볼 수 있다.

참고

http://www.arduino.cc/en/Main/Hardware에서 아두이노 보드에 대한 개요를 볼 수 있다.

http://arduino.cc/en/Guide/Windows(Windows의 경우), http://arduino.cc/en/Guide/MacOSX(Mac OS X의 경우) 및 http://www.arduino.cc/playground/Learning/Linux(Linux의 경우)에서 온라인으로 제공되는 아두이노 가이드를 볼 수 있다.

아두이노 개발 환경과 함께 사용할 수 있는 백여 가지의 보드 목록을 http://jmsarduino.blogspot.com/2009/03/comprehensive-arduino-compatible.html에서 볼 수 있다.

1.1 통합 개발 환경(IDE) 설치하기

과제

아두이노 개발 환경을 컴퓨터에 설치하고 싶다.

해결책

http://arduino.cc/en/Main/Software에서 Windows, Mac 및 Linux용 아두이노 소프트웨어를 다운로드할 수 있다.

Windows용 다운로드 파일은 ZIP 파일이므로 원하는 디렉터리에 파일의 압축을 푼다. 일반적으로 사용되는 경로는 Program Files/Arduino이다.

파일의 압축을 풀어 주는 무료 유틸리티인 7-Zip을 http://www.7-zip.org/에서 다운로드할 수 있다.

파일의 압축을 풀면 Arduino-00〈nn〉이라는 폴더가 생성된다. (여기서 〈nn〉은 다운로드한 아두이노 릴리스의 버전 번호다.) 이 디렉터리에는 실행 파일(Arduino.exe)을 비롯한

여러 개의 파일과 폴더가 있다. Arduino.exe 파일을 두 번 클릭하면 스플래시 화면(그림 1-2)이 표시된 후 기본 프로그램 창이 나타난다(그림 1-3). 소프트웨어가 로드되려면 시간이 걸리므로 조금 기다려야 한다.

그림 1-2 아두이노 스플래시 화면(Windows 7에서 버전 1.0 실행)

Mac용 아두이노 다운로드는 디스크 이미지(.dmg)로 제공되며, 다운로드가 완료된 후 파일을 두 번 클릭한다. 그러면 이미지가 마운트된다. (즉, 데스크톱에 메모리 스틱처럼 표시된다.) 그리고 이 디스크 이미지 안에 아두이노 애플리케이션이 있다. 이제 애플리케이션을 원하는 위치(보통 Applications 폴더 사용)에 복사한다. 그런 다음 애플리케이션을 두 번 클릭하여 실행한다. (애플리케이션을 디스크 이미지에서 직접 실행하는 것은 좋은 방법이 아니다.) 이제 스플래시 화면이 표시된 후 기본 프로그램 창이 표시된다.

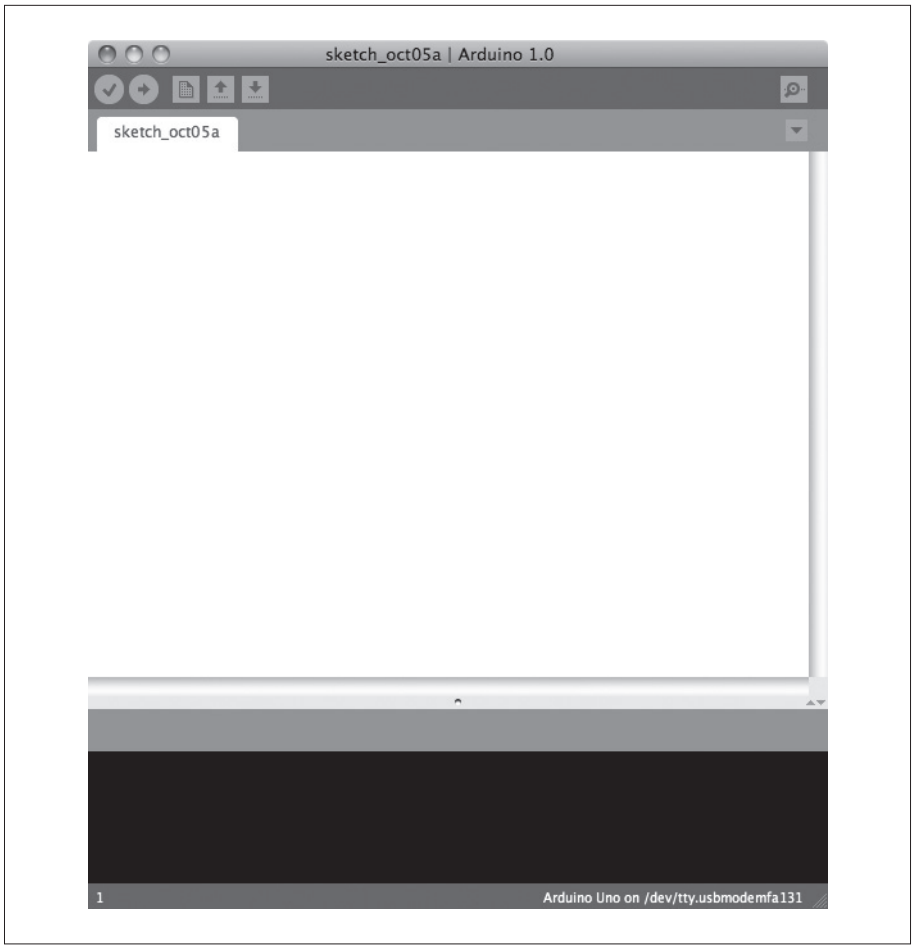


그림 1-3 IDE 기본 창(Mac에서 Arduino 1.0 실행)

Linux 설치는 사용 중인 Linux 배포판에 따라 달라진다. 자세한 정보는 아두이노 위키(http://www.arduino.cc/playground/Learning/Linux)를 참조한다.

아두이노 개발 환경과 보드 간에 데이터를 주고받으려면 드라이버를 설치해야 한다.

Windows의 경우에는 USB 케이블을 사용하여 PC와 아두이노 보드를 연결한 후 새 하드웨어 검색 마법사가 표시될 때까지 기다린다. Uno 보드를 사용 중이라면 마법사를 사용하여 드라이버를 찾아서 설치해 보기 바란다. 아마도 이 작업은 실패할 것이다. (예상된 동작이므로 걱정하지 않아도 된다.) 이 문제를 해결하려면 시작 → 제어

판 → 시스템으로 이동해야 한다. 그런 다음 하드웨어를 클릭하고 장치 관리자를 연다. 표시된 목록에서 Arduino UNO(COM nn)이라는 명칭의 COM 및 LPT 항목을 찾는다. 여기서 nn은 보드용으로 생성된 포트에 자동으로 지정된 번호다. 적합한 드라이버가 아직 지정되지 않았기 때문에 항목 옆에 경고 로고가 표시된다. 이 항목을 마우스 오른쪽 단추로 클릭한 후 드라이버 소프트웨어 업데이트를 선택한다. "컴퓨터에서 드라이버 소프트웨어 찾아보기" 옵션을 선택한 후 조금 전 압축을 풀었던 Arduino 폴더 안에 있는 Drivers 폴더로 이동한다. ArduinoUNO.inf 파일을 선택한다. 그러면 설치 프로세스가 자동으로 완료된다.

이전 버전의 보드를 Windows Vista나 Windows 7과 함께 사용 중이고 온라인 상태라면 마법사를 사용하여 드라이버를 자동으로 검색해서 설치할 수 있다. Windows XP의 경우에는(또는 인터넷이 연결되어 있지 않은 경우) 드라이버의 위치를 지정해 주어야 한다. 파일 선택기를 사용하여 아두이노 파일의 압축을 풀었던 디렉터리 아래에 있는 FTDI USB Drivers 디렉터리로 이동한다. 드라이버 설치가 완료되면 새로운 시리얼 포트가 검색되었다는 메시지를 보여 주는 새 하드웨어 검색 마법사가 다시 표시된다. 이제 앞에서 설명했던 것과 동일한 과정을 수행한다.

한 가지 주의할 점은 드라이버 설치 단계를 2회 반복해야 한다는 것이다. 그렇지 않으면 소프트웨어와 보드 간의 통신이 이루어지지 않는다.

Mac에서는 드라이버를 추가로 설치하지 않고도 Uno와 같은 최신 아두이노 보드를 사용할 수 있다. 보드를 처음 연결하면 새 네트워크 포트가 검색되었다는 알림 메시지가 표시되는데, 이 메시지는 무시해도 된다. 이전 버전의 보드(FTDI 드라이버가 필요한 보드)를 사용 중이라면 드라이버 소프트웨어를 설치해야 한다. 디스크 이미지 안에 FTDIUSBSerialDriver 패키지가 있으며, 이 패키지 이름 뒤에는 버전 번호가 붙어 있다. 이 파일을 두 번 클릭한 후 설치 프로그램의 지시에 따라 설치 프로세스를 진행한다. 설치 프로세스를 완료하려면 관리자 암호를 알고 있어야 한다.

Linux의 경우에는 대부분의 배포판에 드라이버가 이미 설치되어 있다. 하지만 사용 중인 배포판에 해당하는 정보를 보려면 이 장의 소개 절에서 설명한 Linux 링크를 참조하기 바란다.

토론

소프트웨어가 시작되지 않을 경우에는 아두이노 웹 사이트의 문제점 해결 섹션(http://arduino.cc/en/Guide/Troubleshooting)을 참조하여 설치 문제를 해결할 수 있다.

참고

http://arduino.cc/en/Guide/Windows(Windows의 경우), http://arduino.cc/en/Guide/MacOSX(Mac OS X의 경우) 및 http://www.arduino.cc/playground/Learning/Linux(Linux의 경우)에서 온라인으로 제공되는 아두이노 가이드를 볼 수 있다.

1.2 아두이노 보드 설정하기

과제

새 보드에 전원을 공급한 후 보드가 정상적으로 작동하는지 확인하고 싶다.

해결책

보드를 컴퓨터의 USB 포트에 연결한 후 보드에 있는 녹색 LED 전원 표시기가 켜지는지 확인한다. 표준 아두이노 보드(Uno, Duemilanove 및 Mega)의 경우에는 리셋 스위치 근처에 녹색 LED 전원 표시기가 있다.

보드에 전원이 들어오면 보드 가운데에 있는 주황색 LED(그림 1-4의 "핀 13 LED")가 켜졌다가 꺼진다. (보드는 정상 작동 여부를 간단히 검사할 수 있도록 LED를 켰다가 끄는 소프트웨어가 미리 로드된 상태로 공장에서 출고된다.)

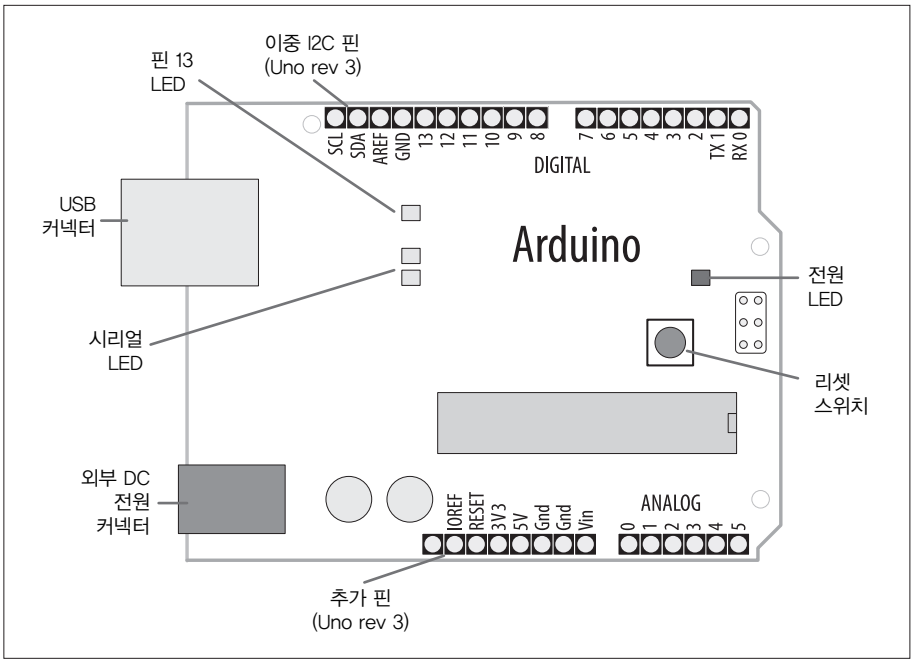


그림 1-4 기본 아두이노 보드(Duemilanove 및 Uno)

Leonardo와 같은 새 보드는 USB 커넥터 옆에 LED가 장착되어 있다(그림 1-5). 최신 보드에는 I2C와 함께 사용할 수 있는 이중 핀이 있다(SCL 및 SDA로 표기). 이들 보드에는 칩의 작동 전압을 결정하는 데 사용할 수 있는 IOREF 핀도 있다.

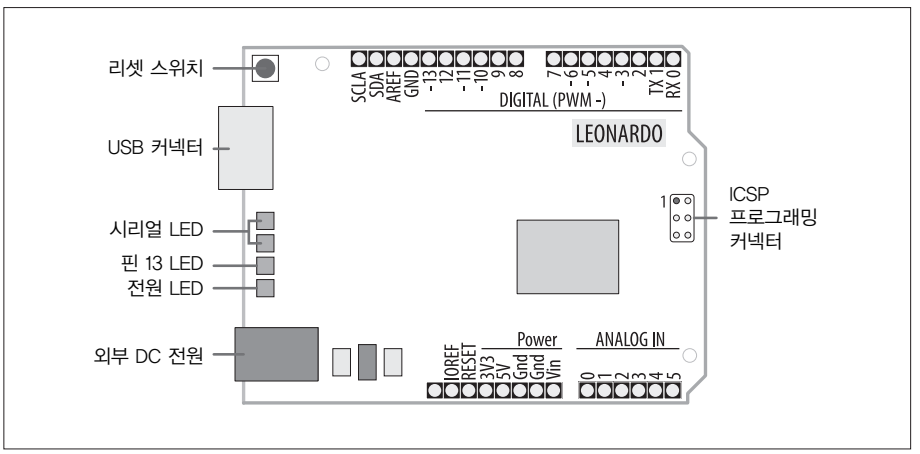
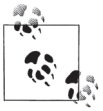


그림 1-5 Leonardo 보드



최신 보드의 경우에는 보드의 커넥터 레이아웃에 대한 새로운 표준에 따라 세 가지 추가 연결 기능이 있다. 하지만 기존 쉴드도 아무 문제 없이 사용할 수 있다. 기존 쉴드는 이전 버전의 보드와 마찬가지로 새 보드에서도 정상적으로 작동한다. 새로 추가된 연결로는 쉴드에서 아날로그 기준 전압을 탐지하여 아날로그 입력값을 공급 전압에 맞춰 조정할 수 있도록 지원하는 IOREF 핀과 I2C 장치의 일관된 연결을 지원하기 위한 SCL 및 SDA 핀이 있다. (이전 버전의 보드에서는 다양한 칩 구성으로 인해 I2C 핀의 위치가 다양했다.) 새 레이아웃에 따라 설계된 쉴드는 새 핀 위치를 사용하는 모든 보드에서 정상적으로 작동해야 한다. IOREF 핀 옆의 추가 핀은 현재 사용되지 않는 핀이지만 향후 핀 레이아웃을 다시 변경하지 않고도 새 기능을 구현할 수 있도록 하기 위해 추가된 것이다.

토론

보드를 컴퓨터에 연결했을 때 전원 LED가 켜지지 않는다면 보드에 전원이 공급되고 있지 않는 것으로 판단할 수 있다.

디지털 출력 핀 13에 연결되어 깜박이는 LED는 보드에서 실행 중인 코드에 의해 제어된다. (새 보드에는 Blink 예제 스케치가 미리 로드되어 있다.) 핀 13 LED가 깜박이고 있다면 스케치가 올바르게 실행 중이라고 간주할 수 있으며, 이는 곧 보드의 칩이 정상적으로 작동하고 있다는 의미이기도 하다. 녹색 전원 LED가 켜져 있기는 하지만 핀 13 LED가 깜박이지 않는다면 팩토리 코드가 칩에 로드되어 있지 않은 경우를 가정할 수 있다. 이 문제가 발생한 경우에는 레시피 1.3의 지침에 따라 Blink 스케치를 보드에 로드하여 보드가 정상적으로 작동하는지 확인한다. 사용하는 보드가 표준 보드가 아닌 경우에는 핀 13에 내장된 LED가 없을 수도 있으므로 해당 보드의 설명서를 참조하기 바란다. Leonardo 보드의 경우에는 정상적으로 작동되는 것을 알려 주기 위해 LED가 서서히 켜졌다 꺼졌다를 반복한다(LED가 "호흡"하는 것처럼 보임).

참고

http://arduino.cc/en/Guide/Windows(Windows의 경우), http://arduino.cc/en/Guide/MacOSX(Mac OS X의 경우) 및 http://www.arduino.cc/playground/Learning/Linux(Linux의 경우)에서 온라인으로 제공되는 아두이노 가이드를 볼 수 있다.

http://arduino.cc/en/Guide/Troubleshooting에서 문제점 해결 가이드를 볼 수 있다.

1.3 통합 개발 환경(IDE)에서 아두이노 스케치 준비하기

과제

보드에 업로드하기 위해 스케치를 가져와서 준비하고 싶다.

해결책

아두이노 IDE를 통해 보드에서 수행할 작업을 정의하는 스케치를 작성하고, 열고, 수정할 수 있다. 이러한 작업은 IDE 상단에 있는 단추(그림 1-6)를 사용하거나 메뉴 또는 단축키(그림 1-7)를 사용하여 수행할 수 있다.

스케치 편집기 영역에서 스케치 코드를 보고 편집할 수 있다. 이 편집기에서는 일반적인 텍스트 편집 키가 지원된다. 예를 들어, Ctrl-F(Mac의 경우 ⌘+F)를 사용하여 텍스트를 검색하거나, Ctrl-Z(Mac의 경우 ⌘+Z)를 사용하여 이전 명령을 실행 취소하거나, Ctrl-C(Mac의 경우 ⌘+C)를 사용하여 선택된 텍스트를 복사하거나, Ctrl-V(Mac의 경우 ⌘+V)를 사용하여 선택된 텍스트를 붙여 넣을 수 있다.

그림 1-7에서는 Blink 스케치(새 아두이노 보드에 미리 로드되어 있는 스케치)를 로드하는 방법을 보여 준다.

IDE를 시작한 후 파일 → 예제 메뉴에서 1. Basics → Blink를 선택한다(그림 1-7). 그러면 내장 LED를 깜박거리게 하는 코드가 스케치 편집기 창에 표시된다(그림 1-6).

코드를 보드로 전송하려면 먼저 아두이노 컨트롤러 칩에서 읽고 실행할 수 있는 명령어로 변환해야 한다. 이 작업을 **컴파일**(compile)이라고 한다. 코드를 컴파일하려면 왼쪽 상단에 체크 표시가 들어 있는 모양의 컴파일 단추를 클릭하거나 스케치 → 확인/컴파일(Ctrl-R, Mac의 경우 ⌘+R)을 선택한다.

이제 텍스트 편집 창 아래의 메시지 영역에 "스케치 컴파일..."이라는 메시지가 표시된다. 잠시 후에 "컴파일 완료"라는 메시지가 표시된다. 검정색 콘솔 영역에 아래와 같은 추가 메시지가 표시된다.

```
바이너리 스케치 사이즈: 1084 바이트(최대 32.256 바이트)
```

이 메시지는 사용 중인 보드와 아두이노 버전에 따라 조금씩 다를 수도 있지만, 해당 스케치의 크기와 보드에서 지원되는 최대 스케치 크기를 알려 준다.

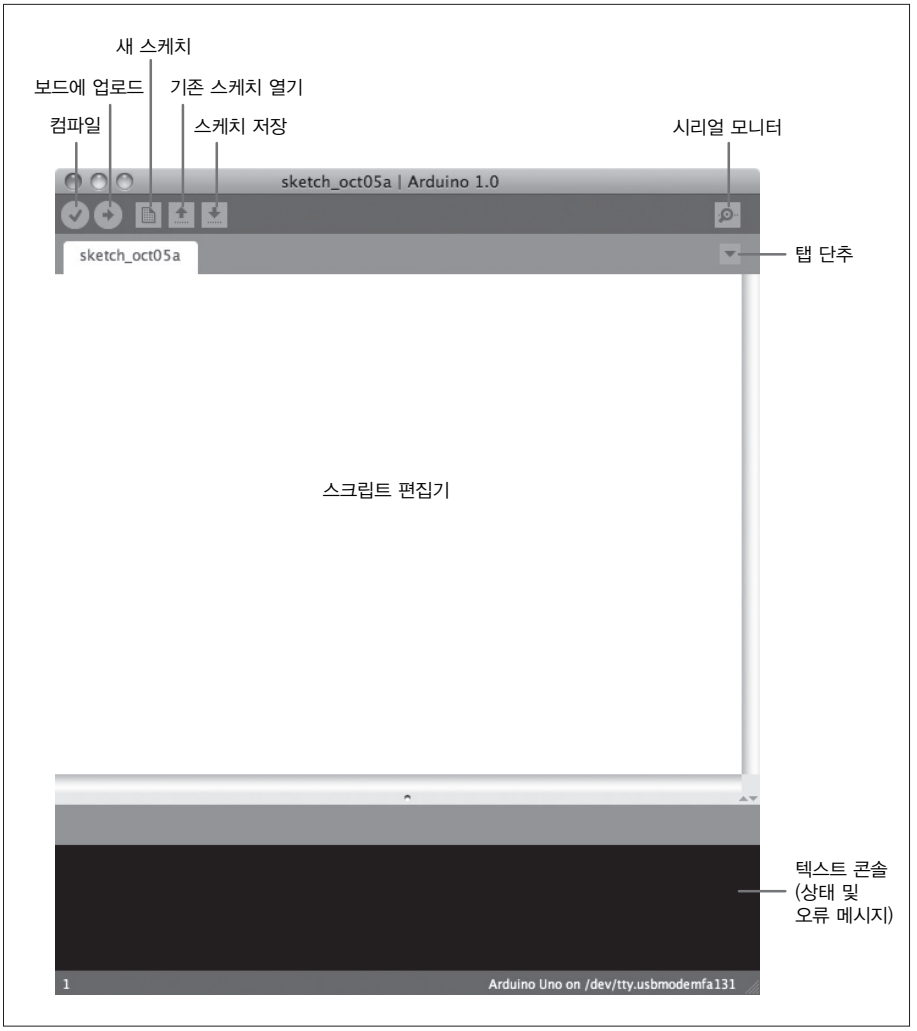


그림 1-6 아두이노 IDE

토론

아두이노에서는 소스 코드를 스케치(sketch)라고 부른다. 그리고 스케치를 가져와서 보드에서 작동되는 형태로 변환하는 프로세스를 컴파일(compile)이라고 한다. 그리고

IDE에서는 보이지는 않지만 수많은 명령줄 도구를 사용하여 스케치를 컴파일한다. 이에 대한 자세한 설명은 레시피 17.1을 참조한다.

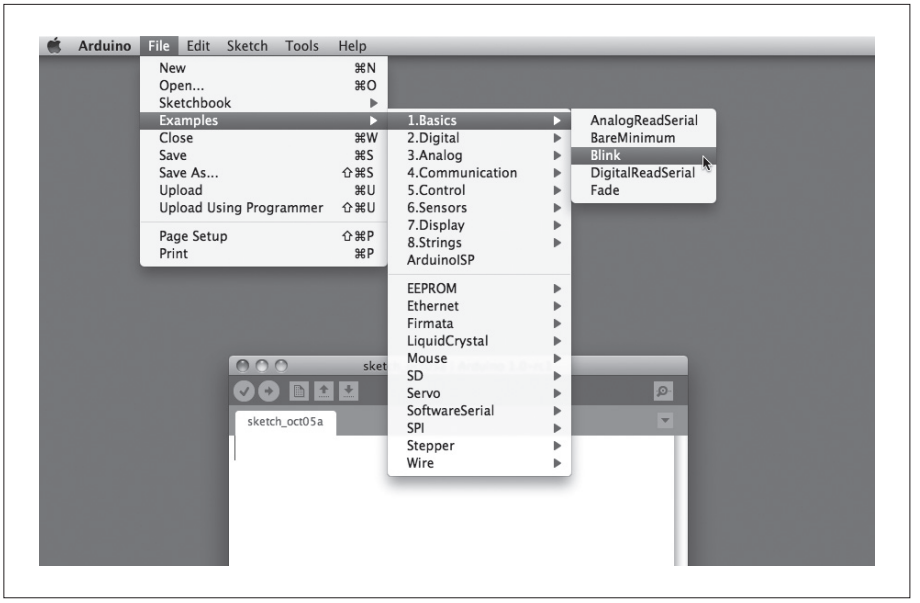


그림 1-7 IDE 메뉴(Blink 예제 스케치 선택)

스케치의 크기를 알려 주는 마지막 메시지에는 컨트롤러 명령어를 보드에 저장하는 데 필요한 프로그램 공간의 용량이 표시된다. 컴파일된 스케치의 크기가 보드의 가용 메모리 용량보다 크면 다음과 같은 오류 메시지가 표시된다.

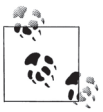
```
Sketch too big; see http://www.arduino.cc/en/Guide/Troubleshooting#size for
 tips on reducing it.
```

이 문제가 발생한 경우에는 보드에 저장할 수 있을 정도로 작게 스케치의 크기를 줄이거나 용량이 큰 보드로 바꿔야 한다.

코드에 오류가 있으면 콘솔 창에 하나 이상의 오류 메시지가 표시된다. 이들 메시지는 오류를 식별하는 데 도움이 된다. 부록 D에서 소프트웨어 오류를 해결하는 데 도움이 되는 정보를 볼 수 있다.

실수로 예제를 덮어쓰는 경우가 발생하지 않도록 하기 위해 아두이노 IDE에서는 제공된 예제 스케치에 변경 사항을 저장하는 기능이 지원되지 않는다. 따라서 다른 이름으로 저장 메뉴 옵션을 사용하여 스케치의 이름을 변경해야 한다. 직접 작성한 스케치는 저장 단추를 사용하여 저장할 수 있다(레시피 1.5).

스케치를 개발하고 수정하다 보면 파일 → 다른 이름으로 저장 메뉴 옵션을 사용하여 이름이나 버전 번호를 정기적으로 변경해 주는 것이 좋다. 이렇게 하면 단계별로 구현해 나가면서 필요에 따라 기존 버전을 쉽게 검토할 수도 있기 때문이다.



보드에 업로드된 코드는 컴퓨터로 다시 다운로드되지 않으므로 스케치 코드를 컴퓨터에 저장해 두는 것을 잊지 말아야 한다. 예제 파일에 변경 사항을 다시 저장할 수는 없으므로 다른 이름으로 저장을 사용하여 변경된 파일을 다른 이름으로 저장해야 한다.

참고

레시피 1.5에서 예제 스케치를 볼 수 있으며, 부록 D에서 소프트웨어 문제 해결에 도움을 주는 팁을 볼 수 있다.

1.4 Blink 스케치 업로드 및 실행하기

과제

컴파일된 스케치를 아두이노 보드로 옮겨서 작동해 보고 싶다.

해결책

USB 케이블을 사용하여 아두이노 보드와 컴퓨터를 연결한다. 레시피 1.3의 설명에 따라 Blink 스케치를 IDE에 로드한다.

그런 다음 드롭다운 메뉴에서 도구 → 보드를 선택하고 연결된 보드의 이름을 선택한다. (연결된 보드가 표준 Uno 보드이면 보드 목록의 첫 번째 항목으로 표시된다.)

이제 도구 → 시리얼 포트를 선택한다. 그러면 컴퓨터에서 사용할 수 있는 시리얼 포

트를 보여 주는 드롭다운 목록이 표시된다. 컴퓨터에서 사용하고 있는 장치에 따라 다양한 조합의 시리얼 포트가 표시된다.

Windows에서는 번호가 매겨진 COM 항목으로 나열된다. 항목이 하나뿐이면 해당 항목을 선택하고, 여러 개의 항목이 있는 경우에는 아마도 마지막 항목이 아두이노 보드에 해당하는 포트일 것이다.

Mac의 경우에는 보드가 두 번 나열되며, Uno 보드일 경우에는 다음과 같이 표시된다.

```
/dev/tty.usbmodem-XXXXXXX
/dev/cu.usbmodem-XXXXXXX
```

구형 보드일 경우에는 다음과 같이 표시된다.

```
/dev/tty.usbserial-XXXXXXX
/dev/cu.usbserial-XXXXXXX
```

XXXXXXX 값은 보드마다 다르다. 이제 두 항목 중 하나를 선택한다.

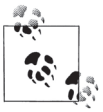
업로드 단추(그림 1-6에서 왼쪽 2번째 단추)를 클릭하거나 파일 → 업로드(Ctrl-U, Mac의 경우 ⌘+U)를 선택한다.

레시피 1.3에서처럼 코드가 컴파일된다. 그리고 컴파일이 완료되면 소프트웨어가 보드에 업로드된다. 이제 보드를 살펴보면 핀 13 LED의 깜박임이 멈추고, 이 LED의 바로 아래에 있는 두 개의 LED(그림 1-4의 시리얼 LED)가 코드가 업로드되는 약 2초 동안 켜졌다가 꺼지는 것을 볼 수 있다. 그런 다음 코드가 실행되면 원래 LED가 다시 깜박이기 시작한다.

토론

IDE에서 컴파일된 코드에 보드로 보내려면 보드가 컴퓨터에 연결되어 있어야 하고 사용하려는 보드와 시리얼 포트를 IDE에서 지정해 주어야 한다.

업로드가 시작되면 보드에서 실행 중인 스케치가 중지된다. (Blink 스케치를 실행 중이었다면 LED의 깜박임이 멈춘다.) 새 스케치는 이전 스케치를 대체하면서 보드에 업로드되며, 업로드가 성공적으로 완료되면 바로 실행된다.



구형 아두이노 보드와 일부 호환 보드에서는 업로드가 시작될 때 실행 중인 스케치가 자동으로 중지되지 않는다. 이 경우에는 소프트웨어의 컴파일이 완료된 직후(스케치 크기에 대한 메시지가 표시된 직후) 보드에 있는 Reset 단추를 눌러야 한다. 컴파일이 완료된 후 Reset 단추를 누를 때 타이밍이 맞지 않으면 이 과정을 다시 반복해야 한다.

업로드가 실패하면 IDE에 오류 메시지가 표시된다. 이 상황은 일반적으로 선택한 보드나 시리얼 포트에 문제가 있거나 보드가 컴퓨터에 연결되어 있지 않은 경우에 발생한다. 아두이노 창 아래쪽 상태 표시줄에 현재 선택된 보드와 시리얼 포트가 표시된다.

Windows에서 올바른 포트가 식별되지 않는 경우에는 보드 연결을 끊은 다음, 도구 → 시리얼 포트를 선택하여 어느 COM 포트가 목록에 표시되지 않는지 확인한다. 또는 보드의 LED가 깜박이면서 코드가 업로드되고 있음을 나타낼 때까지 포트를 하나씩 선택하면서 확인하는 방법을 사용할 수도 있다.

참고

아두이노 문제점 해결 페이지: http://www.arduino.cc/en/Guide/Troubleshooting

1.5 스케치 작성 및 저장하기

과제

스케치를 작성하고 컴퓨터에 저장하고 싶다.

해결책

새 스케치를 작성하기 위해 편집기 창을 열려면 IDE를 실행한 후(레시피 1.3 참조) 파일 메뉴와 새 파일을 차례로 선택한다. 아래 코드를 스케치 편집기 창에 붙여 넣는다. (이 코드는 스케치와 비슷하지만 깜박임 시간이 두 배 더 길다.)

```
const int ledPin = 13;      // LED를 디지털 핀 13에 연결한다.
void setup()
{
  pinMode(ledPin, OUTPUT);
}
```

```
void loop()
{
  digitalWrite(ledPin, HIGH);       // LED를 on으로 설정한다.
  delay(2000);                      // 2초 동안 대기한다.
  digitalWrite(ledPin, LOW);        // LED를 off로 설정한다.
  delay(2000);                      // 2초 동안 대기한다.
}
```

왼쪽 상단에 삼각형이 들어 있는 모양의 컴파일 단추를 클릭하거나 스케치 → 확인/컴파일을 선택하여 코드를 컴파일한다(레시피 1.3 참조).

업로드 단추를 클릭하거나 파일 → 업로드를 선택하여 코드를 업로드한다(레시피 1.4 참조). 업로드가 완료되면 LED가 2초 간격으로 깜박이기 시작한다.

저장 단추를 클릭하거나 파일 → 저장을 선택하여 스케치를 컴퓨터에 저장할 수 있다.

스케치를 새 이름으로 저장하기 위해 다른 이름으로 저장 메뉴 옵션을 선택할 수도 있다. 그러면 파일 이름을 입력할 수 있는 대화 상자가 열린다.

토론

IDE에서 파일을 저장하면 운영 체제의 표준 대화 상자가 열린다. 이 대화 상자에서 My Documents 폴더(Mac의 경우 Documents 폴더)에 있는 Arduino 폴더에 스케치를 저장하는 것이 좋다. 기본 스케치 이름을 스케치의 용도를 설명하는 의미 있는 이름으로 바꿀 수도 있다. 그런 다음 저장을 클릭하여 파일을 저장한다.

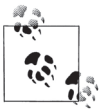
sketch라는 단어 뒤에 현재 날짜가 지정된 이름이 스케치의 기본 이름으로 사용된다. a부터 순서대로 지정되는 문자는 같은 날 작성된 스케치를 구별하는 데 사용된다. 기본 이름을 의미 있는 이름으로 바꾸면 나중에 스케치를 다시 사용하게 될 때 그 용도를 쉽게 알 수 있다.

공백 문자와 같이 IDE에서 지원되지 않는 문자는 자동으로 올바른 문자로 대체된다.

아두이노 스케치는 확장자가 .ino인 일반 텍스트 파일로 저장된다. 이전 버전의 IDE에서는 .pde 확장자를 사용하며, 이는 Processing에서도 사용되는 확장자다. 스케치는 자동으로 같은 이름을 가진 폴더에 저장된다.

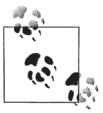
컴퓨터에 있는 어느 폴더에나 스케치를 저장할 수 있지만, 기본 폴더(My Documents

폴더의 Arduino 폴더)에 저장하면 스케치가 아두이노 소프트웨어의 스케치북 메뉴에 자동으로 표시되므로 더 쉽게 찾을 수 있다.



아두이노 다운로드에 포함된 예제를 편집한 경우에는 변경된 파일을 같은 파일 이름으로 저장할 수 없다. 따라서 표준 예제를 변경되지 않은 상태로 유지할 수 있다. 수정된 예제를 저장하려면 다른 위치를 선택해서 스케치를 저장해야 한다.

변경 작업을 완료하게 되면 스케치를 닫을 때 스케치의 저장 여부를 묻는 대화 상자가 표시된다.



스케치 코드를 변경만 하고 저장하지 않으면 IDE 창의 맨 위에 있는 스케치 이름 뒤에 § 기호가 표시된다. 그리고 이 기호는 스케치를 저장하면 사라진다.

아두이노 소프트웨어에는 버전 제어 기능이 없다. 따라서 이전 버전의 스케치로 되돌릴 필요가 있다면, 다른 이름으로 저장 메뉴 옵션을 주기적으로 사용하면서 개정된 각 스케치에 조금씩 다른 이름을 지정해 두는 것이 좋다.

코드를 작성할 때는 코드를 수정하거나 추가할 때마다 컴파일 작업을 수행해서 오류를 자주 검사하는 것이 좋다. 방금 전 작성한 부분에 오류가 있을 확률이 높기 때문에 이렇게 하면 오류를 훨씬 더 쉽게 찾아서 수정할 수 있다.

스케치를 보드에 업로드한 후에는 컴퓨터로 다시 다운로드할 수 있는 방법이 없다. 그러므로 변경된 스케치를 항상 저장해 두는 습관을 가져야 한다.

스케치 파일을 스케치와 다른 이름의 폴더에 저장하려고 하면 파일이 정상적으로 열리지 않을 수도 있으므로 확인을 클릭하여 스케치와 같은 이름의 폴더를 만들도록 권장하는 메시지가 표시된다.

스케치는 같은 이름을 가진 폴더에 있어야 한다. 새 스케치를 저장하면 자동으로 해당 폴더가 생성된다.

이전 버전의 아두이노 소프트웨어로 작성된 스케치는 다른 파일 확장자(.pde)를 사용하지만 아두이노 1.0에서도 열린다. 이전 버전의 스케치를 저장하면 새 확장자(.ino)를 사용하는 파일

이 생성된다. 초기 버전 IDE용으로 작성된 코드는 버전 1.0에서 컴파일되지 않을 수 있다. 이전 버전의 코드를 실행하기 위한 변경 작업은 대부분 쉽게 수행할 수 있다. 자세한 내용은 부록 H를 참조한다.

참고

이 레시피를 비롯한 이 책의 모든 레시피에서는 숫자(13)를 사용하는 대신 `const int` 표현식을 사용하여 의미 있는 이름(`ledPin`)으로 상수를 표현한다. 상수를 사용하는 방법에 대한 자세한 내용은 레시피 17.5를 참조한다.

1.6 아두이노 사용하기

과제

만들기 쉽고 재미 있게 활용할 수 있는 프로젝트부터 시작하고 싶다.

해결책

이 레시피에서는 몇 가지 기술을 간단히 소개한다. 그런 다음 나중에 이후 장에서 자세히 살펴볼 것이다.

이번 레시피에서도 이전 레시피의 LED 깜박임 코드를 기반으로 하지만 이번에는 고정된 지연 시간을 사용하지 않는다. 그 대신 LDR(Light Dependent Resistor)이라는 조명 감지 센서에 의해 깜박임 간격이 결정된다. 그림 1-8과 같이 LDR을 연결한다.

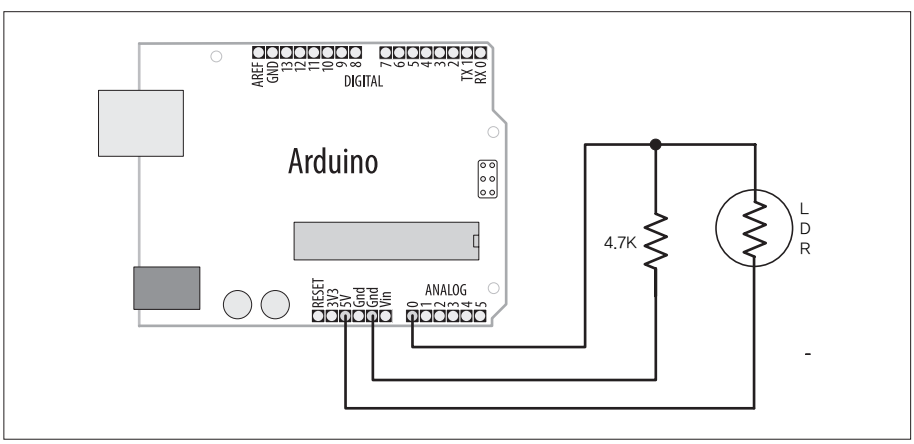
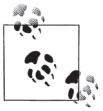


그림 1-8 LDR이 연결된 아두이노



스키마 다이어그램을 보고 회로를 구현하는 데 익숙하지 않다면 부록 B의 단계별 설명을 통해 브레드보드에 회로를 구현하는 방법을 익히기 바란다.

아래 스케치는 아날로그 핀 0에 연결된 LDR의 밝기 레벨을 판독한다. 그런 다음 LDR의 밝기 레벨에 따라 핀 13에 연결된 내부 LED의 깜박임 간격이 결정된다.

```
const int ledPin = 13;      // LED를 디지털 핀 13에 연결한다.
const int sensorPin = 0;    // 센서를 아날로그 핀 0에 연결한다.

void setup()
{
  pinMode(ledPin, OUTPUT);  // LED 핀을 출력으로 설정한다.
}

void loop()
{
  int rate = analogRead(sensorPin); // 아날로그 입력을 읽는다.
  digitalWrite(ledPin, HIGH);       // LED를 on으로 설정한다.
  delay(rate);                      // 밝기 레벨에 따라 지정된 시간 동안 대기한다.
  digitalWrite(ledPin, LOW);        // LED를 off로 설정한다.
  delay(rate);
}
```

토론

4.7K라는 저항 값은 중요하지 않다. 1K부터 10K 사이의 어느 값이나 사용해도 된다. LDR의 밝기 레벨에 따라 아날로그 핀 0의 전압 레벨이 변경된다. analogRead 명령(6장 참조)은 200부터 800 사이의 값을 제공하며, 이 경우 200은 LDR이 어두울 때에 해당하는 값이고, 800은 매우 밝을 때에 해당하는 값이다. 이 값에 따라 LED의 지속 시간이 결정되며, 결과적으로 빛이 밝을수록 깜박임 간격도 증가한다.

다음과 같이 아두이노의 map 함수를 사용하여 깜박임 간격을 확장할 수 있다.

```
const int ledPin = 13;     // LED를 디지털 핀 13에 연결한다.
const int sensorPin = 0;   // 센서를 아날로그 핀 0에 연결한다.

// 다음 두 행에서는 깜박임 간의 최소 및 최대 지연 시간을 설정한다.
const int minDuration = 100;    // 깜박임 간의 최소 대기 시간
const int maxDuration = 1000;   // 깜박임 간의 최대 대기 시간
```

```
void setup()
{
  pinMode(ledPin, OUTPUT);  // LED 핀을 출력으로 설정한다.
}

void loop()
{
  int rate = analogRead(sensorPin);  // 아날로그 입력을 읽는다.
  // 다음 행에서는 최소 및 최댓값 사이의 깜박임 간격을 확장한다.
  rate = map(rate, 200, 800, minDuration, maxDuration);
  // 깜박임 간격으로 변환한다.
  rate = constrain(rate, minDuration, maxDuration);  // 값을 제한한다.
  digitalWrite(ledPin, HIGH);       // LED를 on으로 설정한다.
  delay(rate);                      // 밝기 레벨에 따라 지정된 시간 동안 대기한다.
  digitalWrite(ledPin, LOW);        // LED를 off로 설정한다.
  delay(rate);
}
```

레시피 5.7에서 map 함수를 사용하여 값을 확장하는 방법에 대한 자세한 설명을 볼 수 있다. 레시피 3.5에서는 constrain 함수를 사용하여 값이 지정된 범위를 벗어나지 않도록 제한하는 방법을 볼 수 있다.

컴퓨터에서 rate 변수의 값을 보려면 아래와 같이 루프 코드를 수정하여 아두이노의 시리얼 모니터에 변수 값을 출력할 수 있다. 이 스케치를 실행하면 시리얼 모니터에 깜박임 간격이 표시된다. 아두이노 IDE에서 맨 위 오른쪽에 있는 아이콘을 클릭하면 아두이노 IDE에 시리얼 모니터 창이 표시된다. (4장에서 시리얼 모니터에 대한 자세한 설명을 볼 수 있다.)

```
const int ledPin = 13;      // LED를 디지털 핀 13에 연결한다.
const int sensorPin = 0;    // 센서를 아날로그 핀 0에 연결한다.

// 다음 두 행에서는 깜박임 간의 최소 및 최대 지연 시간을 설정한다.
const int minDuration = 100;        // 깜박임 간의 최소 대기 시간
const int maxDuration = 1000;       // 깜박임 간의 최대 대기 시간

void setup()
{
  pinMode(ledPin, OUTPUT); // LED 핀을 출력으로 설정한다.
  Serial.begin(9600);      // Serial 라이브러리를 초기화한다.
}

void loop()
```

```
{
  int rate = analogRead(sensorPin);  // 아날로그 입력을 읽는다.
  // 다음 행에서는 최소 및 최댓값 사이의 깜박임 간격을 확장한다.
  rate = map(rate, 200, 800, minDuration, maxDuration);
  // 깜박임 간격으로 변환한다.
  rate = constrain(rate, minDuration, maxDuration);   // 값을 제한한다.

  Serial.println(rate);              // rate 값을 시리얼 모니터에 인쇄한다.
  digitalWrite(ledPin, HIGH);        // LED를 on으로 설정한다.
  delay(rate);                       // 밝기 레벨에 따라 지정된 시간 동안 대기한다.
  digitalWrite(ledPin, LOW);         // LED를 off로 설정한다.
  delay(rate);
}
```

그림 1-9처럼 작은 스피커를 핀에 연결한 후 LDR을 사용하여 음의 높낮이를 제어할 수 있다.

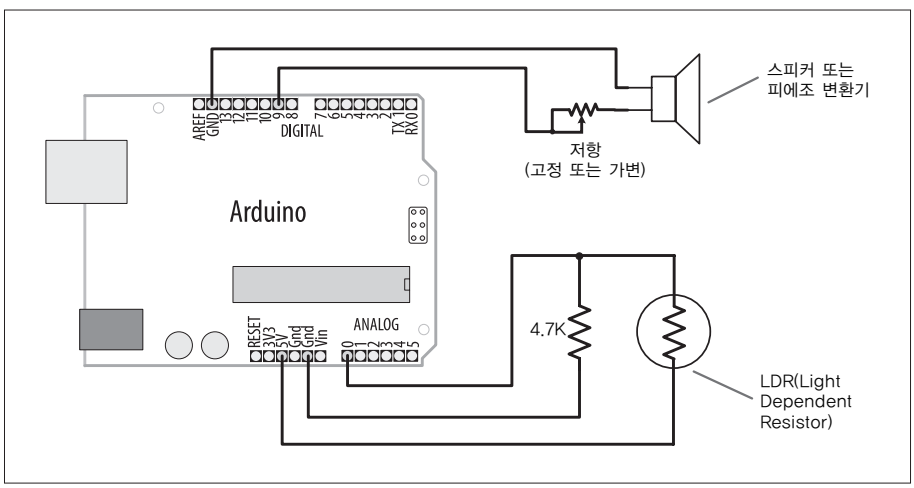


그림 1-9 스피커와 LDR이 연결된 아두이노

이제 핀의 깜박임 간격을 오디오 스펙트럼의 주파수로 조정해야 한다. 이 작업은 아래 코드와 같이 최소 및 최대 기간을 줄이는 방법을 통해 수행할 수 있다.

```
const int outputPin = 9;     // 스피커를 디지털 핀 9에 연결한다.
const int sensorPin = 0;     // 센서를 아날로그 핀 0에 연결한다.

const int minDuration = 1;  // 1ms on, 1ms off(500Hz)
const int maxDuration = 10; // 10ms on, 10ms off(50hz)
```

```
void setup()
{
  pinMode(outputPin, OUTPUT); // LED 핀을 출력으로 설정한다.
}

void loop()
{
  int sensorReading = analogRead(sensorPin);  // 아날로그 입력을 읽는다.
  int rate = map(sensorReading, 200, 800, minDuration, maxDuration);
  rate = constrain(rate, minDuration, maxDuration);  // 값을 제한한다.

  digitalWrite(outputPin, HIGH);    // LED를 on으로 설정한다.
  delay(rate);                      // 밝기 레벨에 따라 지정된 시간 동안 대기한다.
  digitalWrite(outputPin, LOW);     // LED를 off로 설정한다.
  delay(rate);
}
```

참고

constrain 함수를 사용하는 방법에 대한 자세한 설명은 레시피 3.5를 참조한다.

map 함수에 대한 설명은 레시피 5.7을 참조한다.

사운드를 만드는 데 관심이 있다면 아두이노를 사용하여 오디오 출력을 만드는 방법에 대해 자세히 설명하는 9장을 참조한다.