

## Chapter 216. USB 호스트 쉴드

### 1. USB (Universal Serial Bus)

USB는 Universal Serial Bus의 약자로 컴퓨터와 주변기기 사이의 다양한 연결 방식을 통합하여 대체하려는 목적으로 만들어졌다. 1996년 1.0 버전이 출시된 이후 개정을 거듭하여 현재 슈퍼 스피드(Super Speed, SS)라 명명된 3.0 버전까지 발표되었다. USB 3.0은 초당 5 기가비트의 전송 속도를 낼 수 있으며 개발 중인 3.1 버전의 경우 3.0 버전과 비교할 때 2배의 전송속도를 목표로 하고 있는 등 이전 버전의 경우 저속의 장치들에 주로 사용되었던 USB는 고속의 전송을 필요로 하는 디스크 드라이버 등에서도 사용되는 등 적용 범위는 증가하고 있다. 버전에 따른 USB 표준의 최고 속도는 표 1과 같다.

|                  | 속도               | 비고      |
|------------------|------------------|---------|
| Low speed        | 1.5 Mege bit/sec | USB 1.0 |
| Full speed       | 12 Mege bit/sec  |         |
| High speed       | 480 Mege bit/sec | USB 2.0 |
| Super speed      | 5 Giga bit/sec   | USB 3.0 |
| Super speed plus | 10 Giga bit/sec  | USB 3.1 |

표 1. USB 표준 전송 속도

USB는 마스터-슬레이브 구조를 가지고 있으며 호스트, 허브, 디바이스들이 트리 형식으로 연결되는 성형 구조(star topology)를 통해 여러 디바이스들을 연결하고 호스트에 의해 관리된다. 호스트는 USB 연결의 마스터로 USB를 통한 데이터 전송의 모든 권한과 책임을 가진다. 허브는 여러 개의 USB 디바이스들이 하나의 연결을 함께 사용할 수 있도록 해주며 호스트 컨트롤러에 있는 USB 허브를 특히 루트 허브라고 한다. 일반적으로 말하는 호스트는 호스트 컨트롤러와 루트 허브의 조합을 말한다. USB에서는 각 디바이스를 식별하기 위해 7비트의 어드레스를 사용하므로 하나의 호스트에 연결될 수 있는 최대 디바이스의 수는 127개가 된다. 주소 0번은 연결 되었지만 주소가 할당되지 않은 디바이스를 위해 예약된 주소이다. 그림 1은 USB 버스에 호스트를 중심으로 디바이스들이 허브를 통해 성형 구조로 연결된 예를 보여준다. USB는 최대 7개 층(layer)을 지원하므로, 루트 허브를 제외하고 허브는 직렬로 최대 5개

까지 연결될 수 있다.

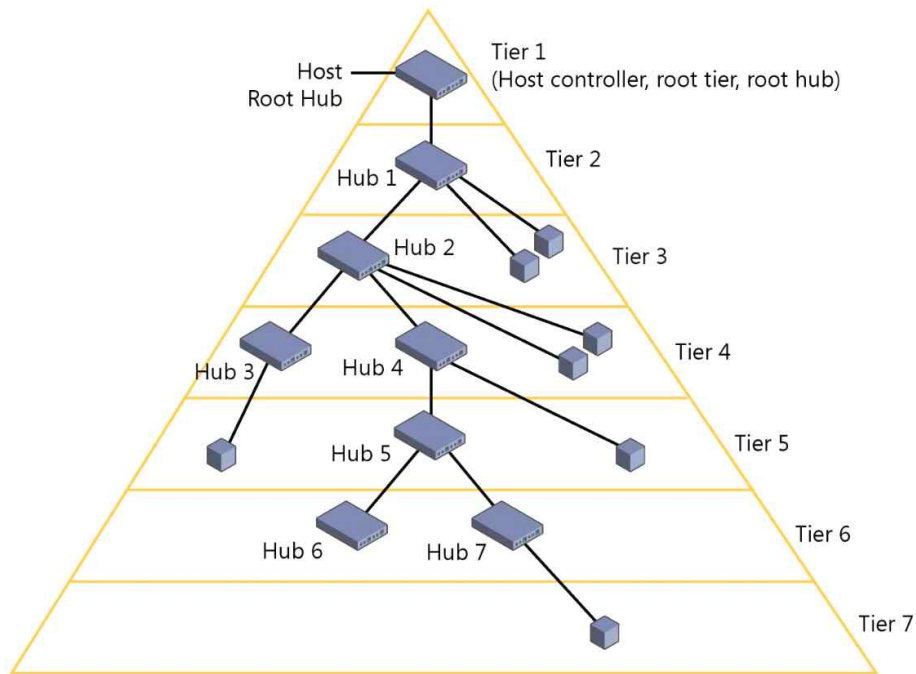


그림 1. USB 연결

USB 호스트 또는 호스트라고도 불리는 장치는 USB를 통한 모든 통신의 책임을 지며, 대표적으로 컴퓨터가 이에 해당한다. 그림 2는 컴퓨터에 USB 허브를 통해 다양한 장치들이 연결된 예를 나타낸 것으로 컴퓨터가 USB 호스트의 역할을 하며 호스트 컨트롤러와 루트 허브로 구성된다. 그림 3은 그림 2를 블록 다이어그램으로 나타낸 것이다.

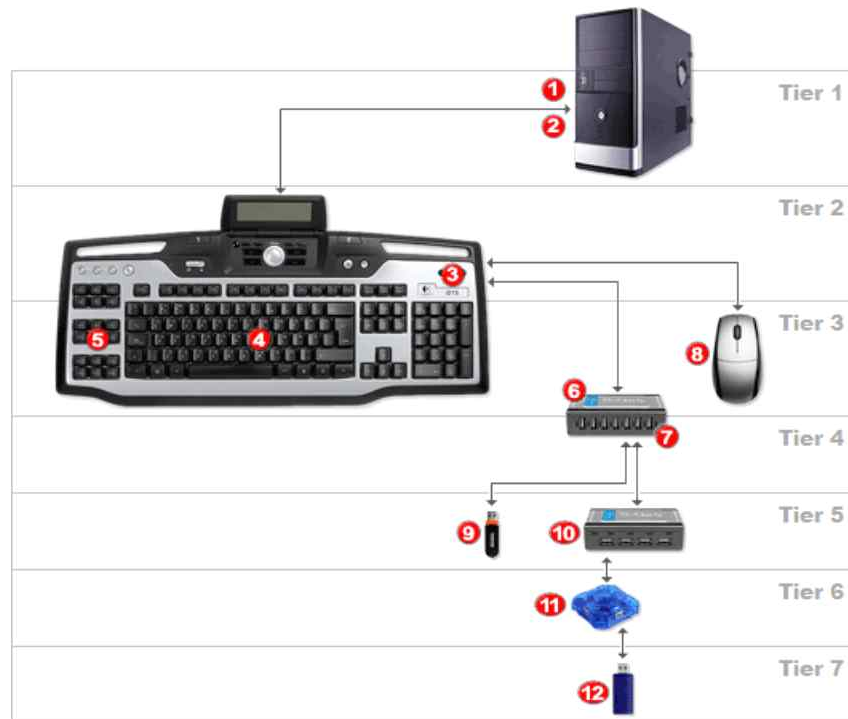


그림 2. PC를 USB 호스트로 한 USB 디바이스 연결

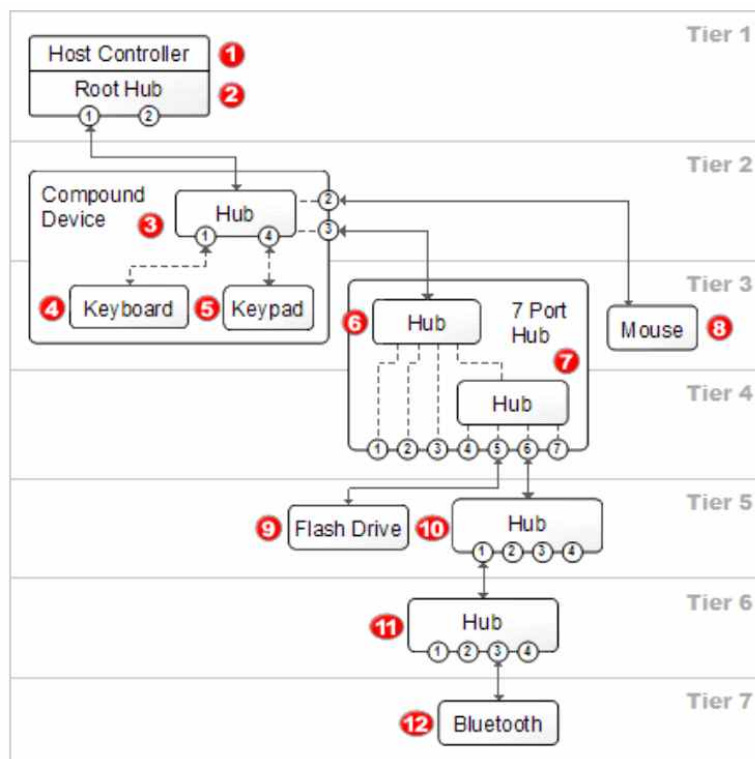
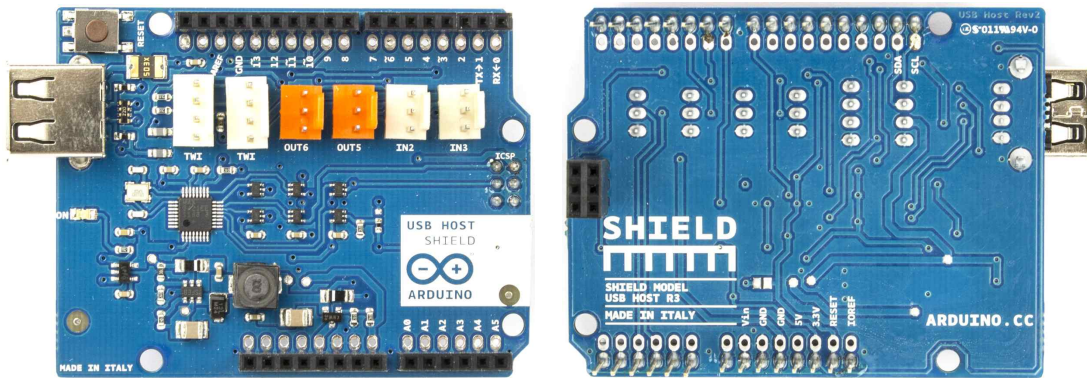


그림 3. 그림 2 USB 연결의 블록 다이어그램

루트 허브는 하나 이상의 USB 연결 단자를 제공하며 기타 허브는 일반적으로 4개나 7개의 연결 단자를 제공한다. 이론적으로 USB 연결에서는 허브를 포함하여 127개의 장치를 연결할 수 있지만 연결된 장치들이 대역폭을 나누어 사용하고 USB를 통해 공급할 수 있는 전력이 3.0 버전에서는 5V, 900mA로 제한되므로 실제 127개의 장치를 연결할 수는 없다.

## 2. USB 호스트 쉴드

아두이노의 공식 쉴드 중 하나인 USB 호스트 쉴드는 아두이노 보드가 USB 호스트로 동작할 수 있도록 해줌으로써 다양한 USB 장치를 손쉽게 아두이노 보드와 연결할 수 있다.



(a) 전면

(b) 후면

그림 4. USB 호스트 쉴드<sup>1)</sup>

USB 호스트 쉴드는 Maxim 사의 MAX3421 칩을 USB 컨트롤러로 사용하고 있다. MAX3421 칩은 USB 2.0 기준에 맞는 풀 스피드 주변장치, 또는 풀 스피드나 로 스피드의 USB 호스트 동작을 지원한다. 컴퓨터의 경우 다양한 USB 장치를 연결할 수 있으며 그 종류는 계속 늘어나고 있다. 하지만 이 모든 장치들을 USB 호스트 쉴드를 통해 아두이노 보드와 연결할 수 있는 것은 아니다. 컴퓨터에 USB 장치를 연결하는 경우 컴퓨터는 호스트의 역할을 수행하기 위해 연결되는 장치에 대한 정보를 알고 있어야 하며 이를 위해 장치 드라이버가 필요하다. 아두이노 보드를 처음 컴퓨터와 연결하였을 때 드라이버가 설치되었던 것을 기억할 것이다. 하지만 아두이노 보드는 컴퓨터와 같이 드라이버를 설치할 수는 없다. 따라서 USB 호스트 쉴드를 통해 아두이노 보드와 연결할 수 있는 장치는 라이브러리에서 지원하는 장치로 한정된다.

1) 이미지 출처: <http://arduino.cc/en/Main/ArduinoUSBHostShield>

하지만 컴퓨터와 비교했을 때 한정된다는 의미이며, USB 호스트 쉴드는 같은 다양한 장치들을 지원하고 있다.

| 종류           | 예                                                          |
|--------------|------------------------------------------------------------|
| HID 장치       | 키보드, 마우스, 조이스틱 등                                           |
| 게임 컨트롤러      | 소니 PS3, 닌텐도 Wii, Xbox 360 등                                |
| USB-시리얼 변환장치 | FTDI, PL-2303, 일부 스마트폰 등                                   |
| 디지털 카메라      | 캐논 EOS, 니콘 DSLR 등                                          |
| 대용량 저장장치     | USB 메모리, 메모리 카드리더, 외장 하드 등                                 |
| 기타           | 구글 ADK (Accessory Development Kit) 지원 스마트폰과 태블릿, 블루투스 동글 등 |

표 2. USB 호스트 쉴드에 연결 가능한 장치

USB 호스트 쉴드를 사용하기 위해서는 먼저 USB 호스트 쉴드 라이브러리를 다운받아<sup>2)</sup> 설치하여야 한다. 라이브러리가 설치되었으면 USB 호스트 쉴드를 아두이노 우노에 적층하고 USB 호스트 쉴드에 USB 마우스를 연결하자.

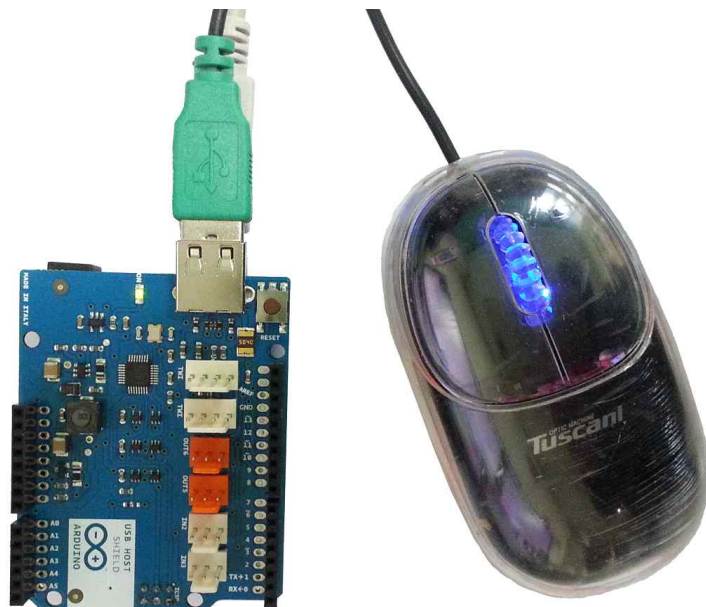


그림 5. USB 호스트 적층 및 마우스 연결

2) [https://github.com/felis/USB\\_Host\\_Shield\\_2.0](https://github.com/felis/USB_Host_Shield_2.0)

USB 호스트 쉴드에 사용된 MAX3421 칩은 SPI 인터페이스를 사용하며 아두이노 보드의 ICSP 커넥터를 통해 연결된다. ICSP 커넥터는 6핀으로 구성되어 있으며 VCC, GND, RESET 이외에 3핀을 USB 호스트 쉴드에서 사용한다. 따라서 SCK(디지털 13번 핀), MISO(디지털 12번 핀), MOSI(디지털 11번 핀)에 해당하는 범용 입출력 핀은 사용할 수 없다.

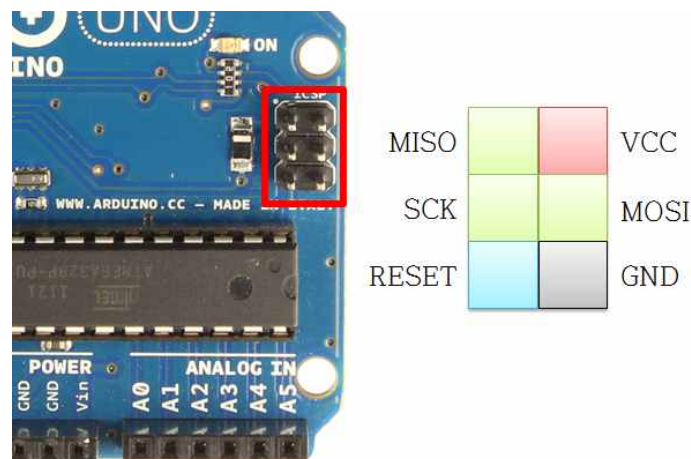


그림 6. 아두이노 우노 보드의 ICSP 커넥터

SCK, MISO, MOSI 핀은 SPI 통신을 사용하는 장치들이 공유하여 사용하지만 장치에 따라 추가로 필요한 전용 핀에 SS 핀이 있다. USB 호스트 쉴드는 SPI의 디폴트 SS핀인 디지털 10번 핀을 사용한다.

하드웨어 준비를 마쳤으면 스케치 1을 입력하여 아두이노 우노에 업로드 하고 마우스를 움직이거나 버튼을 눌러서 시리얼 모니터로 출력되는 메시지를 확인해 보자.

#### 스케치 1. USB 호스트 쉴드 - 마우스 테스트

```
#include <hidboot.h>

// 마우스 동작 처리 클래스
class MouseActionParser : public MouseReportParser
{
protected:
    void OnMouseMove(MOUSEINFO *mi);
    void OnLeftButtonUp(MOUSEINFO *mi);
    void OnLeftButtonDown(MOUSEINFO *mi);
}
```

```
void OnRightButtonUp(MOUSEINFO *mi);
void OnRightButtonDown(MOUSEINFO *mi);
void OnMiddleButtonUp(MOUSEINFO *mi);
void OnMiddleButtonDown(MOUSEINFO *mi);
};

void MouseActionParser::OnMouseMove(MOUSEINFO *mi) // 마우스 이동
{
    Serial.print("dX = ");
    Serial.print(mi->dX, DEC);
    Serial.print(" dY = ");
    Serial.println(mi->dY, DEC);
};

void MouseActionParser::OnLeftButtonUp(MOUSEINFO *mi) // 왼쪽 버튼 떼
{
    Serial.println("Left Button Up");
};

void MouseActionParser::OnLeftButtonDown(MOUSEINFO *mi) // 왼쪽 버튼 누름
{
    Serial.println("Left Button Down");
};

void MouseActionParser::OnRightButtonUp(MOUSEINFO *mi) // 오른쪽 버튼 떼
{
    Serial.println("Right Button Up");
};

void MouseActionParser::OnRightButtonDown(MOUSEINFO *mi) // 오른쪽 버튼 누름
{
    Serial.println("Right Button Down");
};

void MouseActionParser::OnMiddleButtonUp(MOUSEINFO *mi) // 가운데 버튼 누름
{

```

```
    Serial.println("Middle Button Up");
};

void MouseActionParser::OnMiddleButtonDown(MOUSEINFO *mi) // 가운데 버튼 땀
{
    Serial.println("Middle Button Down");
};

USB Usb;                                // USB 연결을 위한 클래스
// USB를 통해 HID 프로토콜을 사용하는 마우스 연결
HIDBoot<HID_PROTOCOL_MOUSE> HidMouse(&Usb);

MouseActionParser parser;

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

    Usb.Init();                          // USB 연결 초기화
    delay(200);

    // 마우스 동작 처리 객체 등록
    HidMouse.SetReportParser(0, (HIDReportParser*)&parser);
}

void loop()
{
    Usb.Task();                          // 마우스 동작 감시
}
```



그림 6. 스케치 1 실행 결과

스케치 1을 테스트하기 위해 사용한 마우스는 HID 프로토콜을 사용하는 마우스이다. HID는 Human Interface Device의 약어로 컴퓨터 사용자가 흔히 사용하는 입력장치의 설치를 간편하기 위해 만들어졌으며 마우스, 키보드, 조이스틱 등이 여기에 포함된다. 스케치 1에서는 마우스 연결을 위해 USB 클래스와 HIDBoot 클래스를 사용하고 있다.

```
USB Usb;
HIDBoot<HID_PROTOCOL_MOUSE> HidMouse(&Usb);
```

USB 클래스는 USB 호스트 칩에서 USB 연결을 관리하기 위한 클래스이며, HIDBoot 클래스는 USB를 통해 연결되는 장치에 대한 클래스이다. 다음으로 필요한 클래스는 MouseReportParser 클래스를 상속하여 만들어지는 사용자 정의 클래스인 MouseActionParser 클래스로 마우스의 동작에 따라 아두이노에서 수행할 작업을 지정하기 위해 사용된다. MouseReportParser 클래스에는 마우스의 움직임과 버튼 클릭에 따라 7개의 가상(virtual) 함수가 선언되어 있으므로 상속을 통해 가상 함수를 구현해 주어야 한다. 7개의 가상 함수를 구현하면 마우스의 움직임이나 버튼을 누르는 동작이 발생하면 자동적으로 호출된다.

```

virtual void OnMouseMove(MOUSEINFO *mi);
virtual void OnLeftButtonUp(MOUSEINFO *mi);
virtual void OnLeftButtonDown(MOUSEINFO *mi);
virtual void OnRightButtonUp(MOUSEINFO *mi);
virtual void OnRightButtonDown(MOUSEINFO *mi);
virtual void OnMiddleButtonUp(MOUSEINFO *mi);
virtual void OnMiddleButtonDown(MOUSEINFO *mi);

```

MouseReportParser 함수에 선언되어 있는 가상 함수는 마우스의 움직임과 버튼 상태에 대한 정보를 가지는 구조체인 MOUSEINFO 형식의 변수를 매개변수로 가진다. 스케치 1에서는 MouseReportParser 클래스를 상속하여 MouseActionParser 클래스를 정의하고 있으며, 각 마우스의 동작에 대해 시리얼 모니터로 메시지를 출력하도록 되어 있다.

마우스를 연결하여 사용하는 과정은 3단계로 이루어진다. 먼저 USB 클래스의 Init 멤버함수를 통해 USB 연결을 초기화하고, HIDBoot 클래스의 SetReportParser 멤버함수를 통해 마우스 이벤트 처리를 담당할 객체를 설정한다. SetReportParser 함수에서는 HIDReportParser 형식의 매개변수를 지정하도록 되어 있으며 HIDReportParser → MouseReportParser → MouseActionParser 순서로 상속이 이루어지므로 사용자 정의 클래스를 형변환을 통해 사용하면 된다. SetReportParser 함수의 첫 번째 매개변수는 터치패드와 마우스가 결합된 경우와 같이 서로 다른 두 종류 이상의 이벤트를 처리할 필요가 있을 때 이를 구별하기 위해 사용하는 ID이다. 실제 마우스 동작은 USB 클래스의 Task 멤버함수를 통해 알아낼 수 있다.

키보드 역시 사용 방법은 비슷하다. 이미 USB 호스트 라이브러리가 설치되어 있으므로 마우스를 제거하고 키보드를 연결하자.



## 그림 7. USB 호스트 쉘드에 키보드 연결

스케치 2를 업로드 하고 키보드 버튼을 눌러 시리얼 모니터로 출력되는 메시지를 확인해 보자.

## 스케치 2. USB 호스트 쉘드 - 키보드 테스트

```
#include <hidboot.h>

// 키보드 동작 처리 클래스
class KeyboardActionParser : public KeyboardReportParser
{
    void PrintKey(uint8_t mod, uint8_t key);

protected:
    void OnKeyDown(uint8_t mod, uint8_t key);
    void OnKeyUp(uint8_t mod, uint8_t key);
};

void KeyboardActionParser::PrintKey(uint8_t m, uint8_t key) // 키 값 화면 출력
{
    MODIFIERKEYS mod;                // 모디파이어 키 상태
    *((uint8_t*)&mod) = m;

    // 왼쪽 모디파이어 키
    Serial.print((mod.bmLeftCtrl == 1) ? "Ctrl " : " ");
    Serial.print((mod.bmLeftShift == 1) ? "Shift " : " ");
    Serial.print((mod.bmLeftAlt == 1) ? "Alt " : " ");

    Serial.print(" << ");
    Serial.print(key);                // 키 값 출력

    // 아스키 문자로 변환 가능한 키가 눌러진 경우 아스키 문자 출력
    uint8_t c = OemToAscii(m, key);   // 아스키 문자로 변환
    if(c) Serial.print(String(", ") + (char)c);
    Serial.print(" >> ");
}
```

```
// 오른쪽 모디파이어 키
Serial.print((mod.bmRightCtrl == 1) ? "Ctrl " : " ");
Serial.print((mod.bmRightShift == 1) ? "Shift" : " ");
Serial.print((mod.bmRightAlt == 1) ? "Alt" : " ");
Serial.println();
};

void KeyboardActionParser::OnKeyDown(uint8_t mod, uint8_t key) // 키 누름
{
    Serial.print("Down : ");
    PrintKey(mod, key);
}

void KeyboardActionParser::OnKeyUp(uint8_t mod, uint8_t key) // 키 떼م
{
    Serial.print("Up   : ");
    PrintKey(mod, key);
}

USB Usb; // USB 연결을 위한 클래스
// USB를 통해 HID 프로토콜을 사용하는 키보드 연결
HIDBoot<HID_PROTOCOL_KEYBOARD> HidKeyboard(&Usb);

KeyboardActionParser parser;

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

    Usb.Init(); // USB 연결 초기화

    delay(200);

    // 키보드 동작 처리 객체 등록
```

```
HidKeyboard.SetReportParser(0, (HIDReportParser*)&parser);  
}  
  
void loop()  
{  
  Usb.Task(); // 키보드 동작 감시  
}
```



그림 8. 스케치 2 실행 결과

마우스의 동작을 처리하기 위해서는 MouseReportParser 클래스를 상속한 사용자 정의 클래스인 MouseActionParser 클래스가 필요한 것처럼, 키보드 동작을 처리하기 위해서는 KeyboardReportParser 클래스를 상속한 사용자 정의 클래스인 KeyboardActionParser 클래스가 필요하다.

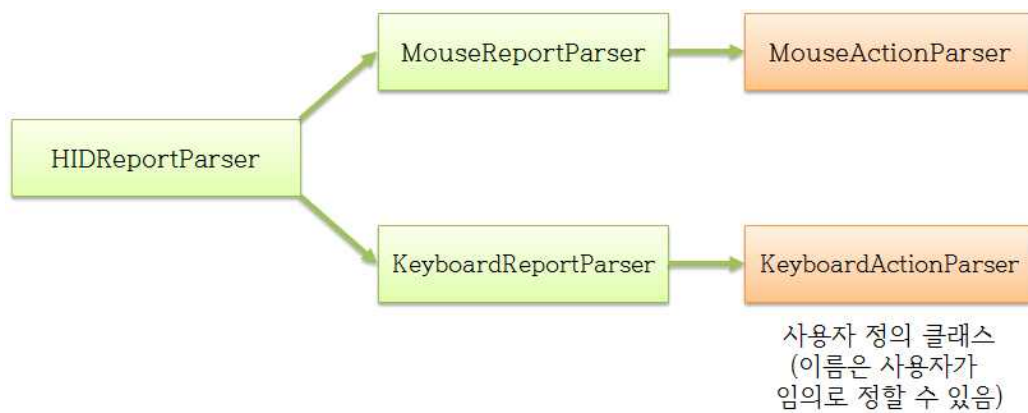


그림 9. 마우스/키보드 동작 처리를 위한 클래스 계층도

KeyboardActionParser 클래스에서는 키를 누른 경우 호출되는 OnKeyDown 멤버함수와 키를 떼는 경우 호출되는 OnKeyUp 멤버함수를 구현하여야 한다.

```

virtual void OnKeyDown(uint8_t mod, uint8_t key);
virtual void OnKeyUp((uint8_t mod, uint8_t key);
  
```

두 멤버함수에서 매개변수 mod는 Shift, Ctrl, Alt 등의 모디파이어 키(modifier key) 상태를 나타낸다. 키보드를 연결하여 사용하는 과정은 마우스와 동일하다. 다만 HIDBoot 클래스의 객체를 생성할 때 HID 프로토콜을 사용하는 키보드임을 명시해주는 것에 차이는 있다.