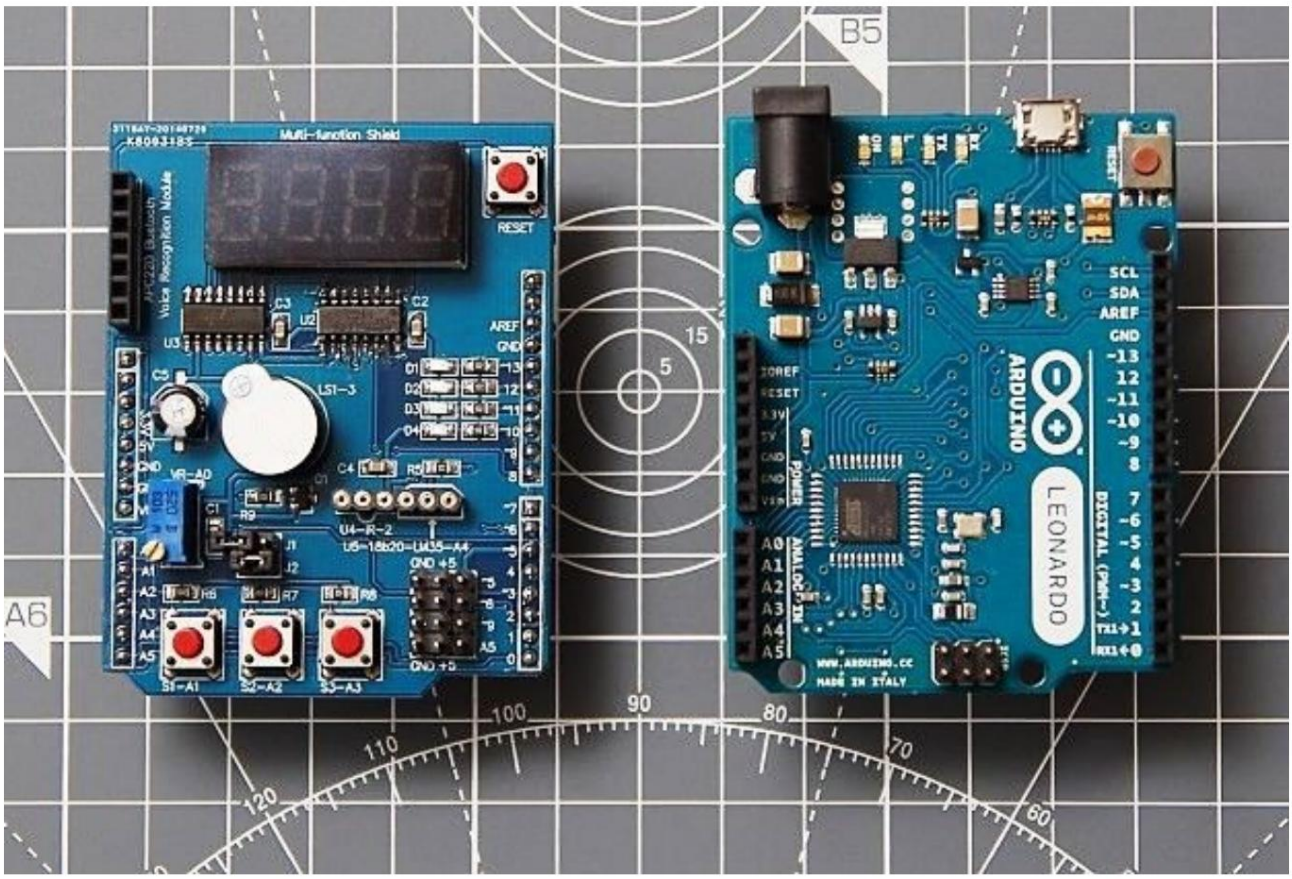
해커트로닉스

Arduino 다기능 실드 프로젝트



작성자: Kashif

Hackatronics 소개 – 재미를 위한 코딩

80년대 초 가정용 컴퓨터 혁명이 시작되는 동안 내 사람들은 나에게 C64 컴퓨터를 사줬고, 얼마 지나지 않아 나는 몇 가지 기본 구성 요소와 센서를 조이스틱과 병렬 포트에 연결하여 흥미로운 작업을 시도하고 수행하는 방법을 배우고 있었습니다. 이것은 1983년으로 거슬러 올라갑니다. 한번은 스위치를 켰을 때 포트에 연결된 전선을 납땜하면서 이 350파운드짜리 컴퓨터를 튀길 수 있었습니다. 다행스럽게도 C64를 최소한의 비용으로 수리했지만 직접 시도하지 마십시오.

오늘날 Raspberry Pi 및 Arduino 마이크로 컨트롤러 제품군은 외부 세계와 연결되는 코드 작성 방법을 배우기 시작하는 좋은 방법입니다. 또한 이미 구축된 구성 요소가 있는 수많은 전자 애드온이 있으며 약간의 코드로 사용할 준비가 되어 있습니다. 그러한 추가 기능 중 하나는 취미 구성 요소와 같은 인터넷 공급업체에서 저렴한 비용으로 사용할 수 있는 다기능 Arduino 실드입니다.

키 누르기 읽기, 디스플레이로 출력하기, 알람 울리기 등과 같이 일반적으로 PC에서 당연하게 여겨지는 간단한 I/O는 마이크로컨트롤러용으로 개발할 때 주요 작업의 초점을 방해하는 경우가 많습니다. 이러한 이유로 저는 기본적이고 일상적인 I/O 작업을 단순화하는 이 다기능 실드용 라이브러리를 개발했습니다. 또한 코딩 시리즈의 일부로 이 라이브러리를 사용하는 실제 응용 프로그램 세트를 제공하므로 Arduino에서 코딩을 처음 접하는 사용자가 이를 실험하고 향상시킬 수 있습니다. 글쎄, 그것이 내가 몇 년 전에 코딩하는 방법을 배운 방법입니다.

Arduino 개발 환경 의 설치와 마찬가지로 Arduino 플랫폼에 대해 어느 정도 익숙하다고 가정 합니다. 데모 비디오 일부 Arduino 응용 프로그램도 사용할 수 있습니다.

이 문서 및 프로젝트의 올바른 URL은 다음과 같습니다.

- https://files.cohesivecomputing.co.uk/Hackatronics-Arduino-Multi-function-Shield.pdf
- https://www.cohesivecomputing.co.uk/hackatronics/

최신 버전의 문서와 소스 코드 샘플을 다운로드했는지 확인하십시오.
코딩 예제는 비상업적 용도로 무료입니다. 어떤 식으로든 도움이 되거나 유용하다고 생각되면 Hackatronics 기금에 소액 기부를 고려하여 흥미로운 프로젝트를 계속 개발할 수 있습니다.

Arduino 다기능 실드 라이브러리 설치

이전에 라이브러리를 설치한 적이 없다면 먼저 Arduino 라이브러리 설치 지침을 검토하십시오.

아래 링크에서 다기능 쉴드 라이브러리를 다운로드하고 위 링크의 지침을 참조하여 .zip 라이브러리로 설치할 수 있습니다.

- 다기능 쉴드 라이브러리

- 시리즈에 사용된 모든 소스 코드

다운로드에 바이러스 및 맬웨어가 없도록 최선을 다하고 있지만 미리 바이러스 및 맬웨어 검사 소프트웨어가 최신 버전인지 확인하십시오.

Hackatronics 시리즈를 따름으로써 귀하는 자신의 책임하에 그렇게 하는 데 동의하고 자신 또는 타인에게 발생할 수 있는 모든 손실 또는 손해에 대해 전적인 책임을 지는 데 동의한다는 점을 유의해야 합니다. 처음 시작하는 어린이라면 책임 있는 성인의 감독을 받아야 합니다.

이 시리즈에 대해 더 알아보기

이 시리즈는 세 가지 주요 부분으로 나뉩니다.

1. 기본 입출력
2. 센서 판독
3. 실제 적용

1부에서는 쉴드 라이브러리를 사용하여 다기능 쉴드 버튼, 신호음 및 디스플레이를 쉽게 사용할 수 있으므로 결과적으로 응용 프로그램의 논리에 더 쉽게 집중할 수 있음을 보여줍니다.

2부에서는 쉴드 라이브러리를 사용하여 온도, 소나 및 모션 센서와 같은 외부 센서의 값을 읽는 방법과 외부 센서의 전자 펄스를 처리하는 방법을 보여줍니다.
원천.

3부에서는 라이브러리와 다기능 실드를 사용하여 작동하는 애플리케이션을 살펴봅니다.

- 24시간 알람 시계
- 심장 모니터 • 카운트다운 타이머 – (심장 맥박 센서 필요)
- 표면 경사도 표시기 – (MPU6050 모션 센서 필요)
- 소나 레인저 – (HC SR04 소나 모듈 필요)
- 속도계 – (자석과 리드 스위치 필요)

이들 각각은 구축하고 확장해야 할 범위가 있지만 그건 여러분에게 맡깁니다.

더 도전적인 프로젝트

우리의 다른 프로젝트를 살펴보십시오. 그 중 일부는 경험이 많은 애호가를 위한 것입니다. 즐거운 코딩하세요!

- Arduino 다기능 쉴드 보너스 스케치
- Arduino 웹 서버 – MVC 디자인을 사용하여 IoT 웹 앱 개발
- Arduino LCD Shield – 코딩 메뉴 쉬운 방법
- Android 및 Arduino 코딩 재미
- Arduino IoT 원격 데이터 수집 및 시각화

1부 기본 입출력

이것은 Arduino Multi-function shield를 위한 Applied Hackatronics 시리즈의 1부로, 차폐 라이브러리를 사용하여 다기능 차폐 버튼, 부저 및 디스플레이에 액세스하는 방법을 보여줍니다.
아직 설치하지 않았다면 소스 코드를 다운로드하고 소개에 있는 링크를 사용하여 라이브러리를 설치해야 합니다.

Hackatronics 시리즈를 따름으로써 귀하는 자신의 책임하에 그렇게 하는 데 동의하고 귀하 또는 다른 사람에게 발생할 수 있는 모든 손실 또는 손해에 대해 전적인 책임을 지는 데 동의하는 것입니다.

방패의 비퍼를 사용하여

다기능 실드 라이브러리는 신호음을 사용하여 다양한 유형의 경보를 울리는 유연한 방법을 제공합니다. 신호음의 실제 타이밍과 소리는 인터럽트를 사용하여 백그라운드에서 제어되므로 애플리케이션이 계속해서 주요 작업을 수행하는 데 집중할 수 있습니다.

```
#include <MultiFuncShield.h>

무효 설정() {
    // 여기에 설정 코드를 넣어 한 번 실행합니다.

    MFS.initialize();                       // 다기능 쉴드 라이브러리 초기화

    // 참고 신호음 제어는 백그라운드에서 수행됩니다. 즉, beep()은 차단되지 않습니다.


    // 200밀리초 동안 짧은 경고음
    MFS.beep();

    지연(1000);

    // 4번의 짧은 비프음, 3번 반복.
    MFS.beep(5, 5, //           // 50밀리초 동안 신호음
                    50밀리초 동안 무음
                    4, // 위의 사이클을 4번 반복합니다.
                    3, // 3번 반복
                    50          // 루프 사이에 500밀리초를 기다립니다.
                    );
}

무효 루프() {
    // 여기에 기본 코드를 넣어 반복적으로 실행합니다.

}
```

실드의 버튼 누름 감지

다기능 실드 라이브러리를 사용하면 짧게 누르기, 길게 누르기, 짧게 누른 후 버튼 해제, 길게 누른 후 버튼 해제 등 다양한 유형의 버튼 누름을 감지할 수 있습니다. 아래 스케치는 직렬 모니터 창에서 버튼 누르기 유형을 표시합니다. 여러 버튼을 함께 누르고 있거나 다른 기간 동안 어떤 일이 발생하는지 확인하십시오.

```
#include <MultiFuncShield.h>

무효 설정() {
    // 여기에 설정 코드를 넣어 한 번 실행합니다.
    Serial.begin(9600);
    MFS.initialize();                       // 다기능 쉴드 라이브러리 초기화
}
```

```
void loop() { // 여기에 메
    인 코드를 넣어 반복적으로 실행합니다.

    바이트 btn = MFS.getButton(); // 일반적으로 반환값을 비교하는 것으로 충분합니다.
                                                    // 미리 정의된 매크로에 대한 값, 예
BUTTON_1_PRESSED,
                                                    // BUTTON_1_LONG_PRESSED 등

    만약 (btn) {

        바이트 버튼 번호 = btn & B00111111; 바이트 buttonAction = btn &
        B11000000;

        Serial.print("BUTTON_"); Serial.write(버
        튼 번호 + '0'); Serial.print("_");


        if (buttonAction == BUTTON_PRESSED_IND) {

            Serial.println("눌림");

        } else if (buttonAction == BUTTON_SHORT_RELEASE_IND) {

            Serial.println("SHORT_RELEASE");

        } else if (buttonAction == BUTTON_LONG_PRESSED_IND) {

            Serial.println("LONG_PRESSED");

        } else if (buttonAction == BUTTON_LONG_RELEASE_IND) {

            Serial.println("LONG_RELEASE");
        }
    }
}
```

실드의 디지트 디스플레이에 쓰기 다기능 실드의

디지트 디스플레이 관리는 인터럽트를 사용하여 백그라운드에서 수행되며, 이는 애플리케이션이 계속해서 주요 작업을 수행하는 데 집중할 수 있음을 의미합니다. 문자열, 정수 및 부동 소수점 값은 아래 스케치와 같이 디스플레이에 기록됩니다.

```
#include <MultiFuncShield.h>

무효 설정() {
    // 여기에 설정 코드를 넣어 한 번 실행합니다.

    MFS.initialize();                    // 다기능 쉴드 라이브러리 초기화

    MFS.write("안녕하세요"); 지연
    (2000); MFS.write(-273); 지연
    (2000); MFS.write(3.141,
    2); // 소수점 이하 2자리까지 표
    시합니다. 지연(2000);

}

정수 카운터 = 0; 바이트 종
료 = 거짓;

void loop() { // 여기에 메
    인 코드를 넣어 반복적으로 실행합니다.

    if (카운터 < 200) {

        MFS.write((int)카운터); 카운터++;


    } else if(!종료) {
```

```
            종료 = 참; MFS.write("종
            료"); MFS.blinkDisplay(DIGIT_ALL,
            ON);

        } 지연(50);
}
```

쉴드의 LED 조명 제어하기 쉴드의 LED

조명 을 제어하기 위해 반드시 다기능 쉴드 라이브러리를 사용할 필요는 없지만 애플리케이션에서 기본적인 깜박임 작업을 수행하기 위해 LED가 필요한 경우 지원이 제공됩니다. 깜박임은 인터럽트를 사용하여 백그라운드에서 관리됩니다.

```
#include <MultiFuncShield.h>

무효 설정() {
    // 여기에 설정 코드를 넣어 한 번 실행합니다.

    MFS.initialize();                    // 다기능 쉴드 라이브러리 초기화

    MFS.writeLeds(LED_ALL, ON); 지연(2000);
    MFS.blinkLeds(LED_1 | LED_2, ON); 지연(2000);
    MFS.blinkLeds(LED_1 | LED_2, OFF);



    MFS.blinkLeds(LED_3 | LED_4, ON); 지연(2000);
    MFS.blinkLeds(LED_ALL, ON); 지연(2000);
    MFS.blinkLeds(LED_ALL, OFF); MFS.writeLeds(LED_ALL,
    OFF);


}

void loop() { // 여기에 메
    인 코드를 넣어 반복적으로 실행합니다.

}
```

쉴드의 전위차계 값 읽기 이 스케치는 미리 설정된 팟의 값

을 읽고 다기능 쉴드에 표시하는 방법을 보여줍니다. 이 스케치를 업로드한 후 전위차계의 나사를 돌려 숫자 디스플레이에서 판독값 변화를 확인합니다.

```
#include <MultiFuncShield.h>

무효 설정() {
    // 여기에 설정 코드를 넣어 한 번 실행합니다.

    MFS.initialize();                    // 다기능 쉴드 라이브러리 초기화
}

void loop() { // 여기에 메
    인 코드를 넣어 반복적으로 실행합니다.

    MFS.write(analogRead(POT_PIN));

    지연(100);
}
```

모든 코드 샘플과 애플리케이션은 테스트를 거쳤으며 작동합니다. 어려움이 있는 경우 댓글을 남겨주시면 최대한 빨리 답변해 드리겠습니다.

2부 센서 판독

이것은 Arduino 다기능 실드를 위한 Applied Hackatronics 시리즈의 2부이며 다기능 실드 라이브러리를 사용하여 온도, 소나 및 모션 센서와 같은 외부 센서의 값을 읽는 방법과 전자 처리 방법을 보여줍니다. 외부 소스의 펄스. 아직 설치하지 않았다면 소스 코드를 다운로드하고 소개에 있는 링크를 사용하여 라이브러리를 설치해야 합니다.

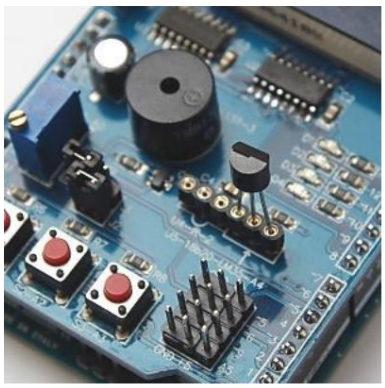
Hackatronics 시리즈를 따름으로써 귀하는 자신의 책임하에 그렇게 하는 데 동의하고 귀하 또는 다른 사람에게 발생할 수 있는 모든 손실 또는 손해에 대해 전적인 책임을 지는 데 동의하는 것입니다.

펄스 계산

다기능 쉴드 라이브러리는 Arduino의 입력 핀에 적용된 펄스(최대 500Hz)를 카운팅하는 기능을 지원합니다. 펄스 계산은 인터럽트를 사용하여 백그라운드에서 관리되므로 애플리케이션이 주요 작업을 수행하는 데 집중할 수 있습니다. 이 스케치를 업로드한 후 반복적으로
버튼 1을 눌러 펄스를 생성하고 디지트 디스플레이에서 프레스 속도 판독값을 확인합니다.

```
#include <MultiFuncShield.h>

무효 설정() {
    // 여기에 설정 코드를 넣어 한 번 실행합니다.

    MFS.initialize(); // 다기능 쉴드 라이브러리 초기화

    MFS.initPulseInCounter(
            BUTTON_1_PIN,                   // 펄스 생성 수단으로 버튼 1을 사용합니다.
            1500, // 펄스를 기다리는 시간(밀리초)
주기의 펄스를 0으로 재설정합니다.
            낮은                                    // LOW 입력에서 펄스를 트리거합니다.
    );
}


무효 루프() {
    // 여기에 기본 코드를 넣어 반복적으로 실행합니다.

    // 가장 최근 펄스의 주기를 가져옵니다(밀리초).
    // 참고: 펄스 측정은 실제로 인터럽트를 사용하여 수행됩니다.
    부호 없는 정수 pulsePeriodMs = MFS.getPulseInPeriod();

    if (펄스 주기 == 0)
    {
        MFS.write(0.0, 1);
    }
    또 다른
    {
        MFS.write(1000.0 / pulsePeriodMs, 1); // 초당 펄스를 계산합니다. 표시하다
소수점 1자리까지.
    }
}
```

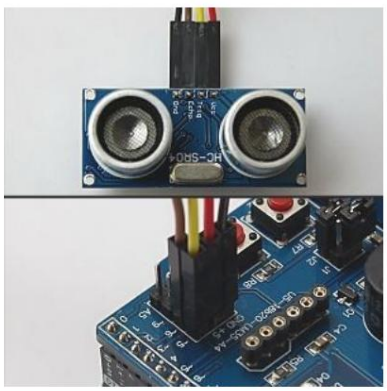
LM35 센서를 사용하여 온도 읽기



다기능 쉴드에는 LM35 온도 센서를 수용하기 위한 소켓이 있습니다. 이 소켓은 올바르게 삽입해야 합니다. 그렇지 않으면 Arduino 또는 연결된 컴퓨터가 복구할 수 없을 정도로 손상될 수 있습니다.
센서가 잘못 연결되면 매우 뜨거워지기 때문에 알 수 있습니다. 다기능 실드 라이브러리는 이 센서의 판독값을 평활화하기 위해 세 가지 수준의 필터링을 제공합니다.

```
#include <MultiFuncShield.h>

// 참고: 점퍼 J1이 실드에서 제거되었고 LM35가 올바르게 삽입되었는지 확인하십시오.


무효 설정() {
    // 여기에 설정 코드를 넣어 한 번 실행합니다.
    MFS.initialize(); // 다기능 쉴드 라이브러리 초기화

    // 저역 통과 필터를 사용하여 초기화합니다.
    // SMOOTHING_NONE, SMOOTHING_MODERATE 또는 SMOOTHING_STRONG 중 하나를 선택합니다.
    MFS.initLM35(SMOOTHING_MODERATE);
}

무효 루프() {
    // 여기에 기본 코드를 넣어 반복적으로 실행합니다.

    정수 tempCentigrade = MFS.getLM35Data(); // 섭씨 1/10도를 얻습니다.

    MFS.write((float)tempCentigrade / 10, 1); // 소수점 이하 1자리까지 온도를 표시합니다.

    지연(100);
}
```

HC SR04 소나 모듈 사용

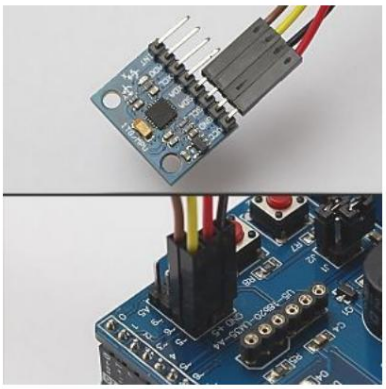


HC SR04 소나 모듈을 사용할 때 인터럽트가 켜져 있는 경우 다기능 실드 라이브러리를 사용하여 거리 값을 읽고 계산하는 것이 좋습니다. 그 이유는 라이브러리의 인터럽트 서비스 루틴이 이 소나의 타이밍 요구 사항에 영향을 미치기 때문입니다.
모듈이며 라이브러리가 이를 보상합니다. 라이브러리는 또한 소나 모듈의 판독값을 부드럽게 하기 위해 세 가지 수준의 필터링을 제공합니다. 소나 모듈의 트리거 및 에코 핀은 각각 Arduino 핀 5 및 6에 연결됩니다.

```
#include <MultiFuncShield.h>

const int TrigPin = 5;
const int 에코핀 = 6;


무효 설정() {
    // 여기에 설정 코드를 넣어 한 번 실행합니다.
```

```
    핀모드(TrigPin, 출력); 핀모드(에코핀, 입력);


    MFS.initialize(); // 다기능 쉴드 라이브러리 초기화

    // 저역 통과 필터로 초기화: SMOOTHING_NONE, SMOOTHING_MODERATE 또는
SMOOTHING_STRONG
    MFS.initSonar(SMOOTHING_MODERATE);
}


void loop() { // 여기에 메
    인 코드를 넣어 반복적으로 실행합니다.

    MFS.write((int)MFS.getSonarDataCm(TrigPin, EchoPin));

    지연(100);
}
```

MPU6050 모션 센서에서 데이터 가져오기



다기능 실드는 Leonardo와 같은 일부 Arduino 보드의 하드웨어 I2C 핀을 노출하지 않습니다. 다음 스케치는 실드 라이브러리를 사용하여 와이어 라이브러리를 사용하여 MPU6050 센서에서 원시 값을 읽습니다. 참고: 옆의 이미지는 더 이상 지원되지 않는 소프트웨어 I2C를 사용할 때의 핀 연결을 보여줍니다.

스케치를 Arduino에 업로드하려면 전체 소스 코드를 다운로드하십시오. 그런 다음 실드의 버튼 1을 눌러 센서 데이터 보고를 토글하고 버튼 2를 눌러 가속도, 자이로 및 온도 값을 순환합니다.

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

#"I2C.h" 포함 #"MPU6050.h"
포함

MPU6050 MPU; 무효
displayHeading(바이트 모드);


무효 설정()
    { Serial.begin(9600);

    // 여기에 설정 코드를 넣어 한 번 실행합니다. // 하드웨어 사용 I2C Wire.begin();
    I2C1.initialize(&Wire); MPU.initialize(&I2C1, MPU_DEFAULT_ADDRESS);




    MFS.initialize(); MFS.write("계
    정");
}


바이트 displayValues = true; 바이트 표시 모드 = 0;



void loop() { // 여기에 메
    인 코드를 넣어 반복적으로 실행합니다.
```

```
    바이트 btn = MFS.getButton();

    // 버튼 1을 사용하여 센서 값 보고를 토글합니다. if (btn == BUTTON_1_PRESSED) {


        디스플레이 값 = !디스플레이 값;

        if (디스플레이 값) {

            displayHeading(디스플레이 모드);

        } 또 다른
        {
            MFS.write("꺼짐");
        }
    }

    if (디스플레이 값) {

        // 버튼 2를 사용하여 디스플레이 모드를 순환합니다. if (btn == BUTTON_2_PRESSED) {


            디스플레이 모드++;

            if (디스플레이 모드 == 3) {

                디스플레이 모드 = 0;
            }

            displayHeading(디스플레이 모드);
        }

        if (디스플레이 모드 == 0) {

            // 원시 가속 값을 표시합니다.
            MPU.getAccelRaw();
            Serial.print((float)MPU.accel_X_Raw / MPU.accelScaleValue); Serial.print("\t");
            Serial.print((float)MPU.accel_Y_Raw / MPU.accelScaleValue); Serial.print("\t");
            Serial.print((float)MPU.accel_Z_Raw / MPU.accelScaleValue); Serial.print("\t\n");




        } else if (디스플레이 모드 == 1) {

            // 원시 자이로값 표시 MPU.getGyroRaw();
            Serial.print((float)MPU.gyro_X_Raw /
            MPU.gyroScaleValue); Serial.print("\t"); Serial.print((float)MPU.gyro_Y_Raw / MPU.gyroScaleValue);
            Serial.print("\t"); Serial.print((float)MPU.gyro_Z_Raw / MPU.gyroScaleValue); Serial.print("\t\n");




        } else if (디스플레이 모드 == 2) {

            // 온도 값을 표시합니다.
            Serial.println((float)MPU.getTemp10th() / 10);
        }
    }

    지연(50);
}


무효 displayHeading(바이트 모드) {

    if (모드 == 0) {

        Serial.println("가속도 g(1g = 9.8 m/s/s)"); Serial.println("X\tY\tZ");
```

```
            MFS.write("계정");

        } else if (모드 == 1) {

            Serial.println("자이로 각속도(단위: 도/초)"); Serial.println("X\tY\tZ"); MFS.write("자이로");



        } else if (모드 == 2) {

            Serial.println("섭씨 온도입니다."); MFS.write("테");

        }
    }
```

모든 코드 샘플과 애플리케이션은 테스트를 거쳤으며 작동합니다. 어려움이 있는 경우 댓글을 남겨주시면 최대한 빨리 답변해 드리겠습니다.

3부 실제 적용

이것은 Arduino 다기능 실드용 응용 Hackatronics 시리즈의 3부이며 라이브러리와 다기능 실드를 사용하여 작동하는 응용 프로그램을 탐색합니다. 아직 하지 않았다면
그렇게 하면 소스 코드를 다운로드하고 소개에 있는 링크를 사용하여 라이브러리를 설치해야 합니다.

아래의 각 응용 프로그램에 대해 온라인 비디오 가 함께 제공됩니다. 요약 동영상도 포함됩니다.

Hackatronics 시리즈를 따름으로써 귀하는 자신의 위험을 감수하는 데 동의하고 자신이나 다른 사람에게 발생할 수 있는 모든 손실이나 손해에 대해 전적인 책임을 지는 데 동의하는 것입니다.

카운트 다운 타이머

이 카운트다운 타이머는 전자레인지에서 볼 수 있는 카운트다운 타이머와 유사합니다. 당신은 설정
시간, 카운트다운 타이머를 시작하고 0에 도달하면 알람이 울립니다. 타이머를 일시 중지/계속하고 0으로
재설정할 수 있습니다. 다기능 실드 버튼 2와 3을 사용하여 분과 초를 설정합니다. 버튼 1을 짧게 누르면 타이머가 시작되거나 중지되고 길게 누르면 재설정됩니다.
이 응용 프로그램의 가능한 개선 사항은 타이머가 카운트다운되는 동안에만 장치를 켜는 것입니다.

```
#include <MultiFuncShield.h>


열거형 CountDownModeValues
{
    COUNTING_STOPPED,
    계산
};

바이트 countDownMode = COUNTING_STOPPED;

10분의 1바이트 = 0;
문자 초 = 0;
문자 분 = 0;

무효 설정() {
    // 여기에 설정 코드를 넣어 한 번 실행합니다.
    MFS.initialize(); MFS.write(0);                // 다기능 쉴드 라이브러리 초기화


    Serial.begin(9600);
}


무효 루프() {
    // 여기에 기본 코드를 넣어 반복적으로 실행합니다.

    바이트 btn = MFS.getButton();

    스위치(카운트다운 모드)
    {
        사례 COUNTING_STOPPED:
            if (btn == BUTTON_1_SHORT_RELEASE && (분 + 초) > 0)
            {
                // 타이머 시작
                countDownMode = 계산;
            }
            그렇지 않으면 (btn == BUTTON_1_LONG_PRESSED)
            {
```

```
                    // 타이머 10분의 1을 재설정 = 0; 초 = 0;
                    분 = 0; MFS.write(분*100 + 초);



                } else if (btn == BUTTON_2_PRESSED || btn == BUTTON_2_LONG_PRESSED) {

                    분++; if (분 > 60) {


                        분 = 0;
                    }
                    MFS.write(분*100 + 초);

                } else if (btn == BUTTON_3_PRESSED || btn == BUTTON_3_LONG_PRESSED) {

                    초 += 10; if (초 >= 60) {


                        초 = 0;
                    }
                    MFS.write(분*100 + 초);

                } 부서지다;

            사례 계산:
                if (btn == BUTTON_1_SHORT_RELEASE || btn == BUTTON_1_LONG_RELEASE) {

                    // 타이머 중지 countDownMode =
                    COUNTING_STOPPED;
                }
                또 다른
                {
                    // 계속해서 10분의 1까지 카운트다운++;


                    if (십분의 일 == 10) {

                        십분의 일 = 0; 초--;


                        if (초 < 0 && 분 > 0) {

                            초 = 59; 분--;

                        }

                        if (분 == 0 && 초 == 0) {

                            // 타이머가 0에 도달했으므로 알람을 울립니다. MFS.beep(50, 50, 3); // 비프음 3번, 500밀리초
                            켜짐 / 500꺼짐 countDownMode = COUNTING_STOPPED;

                        }

                        MFS.write(분*100 + 초);

                    } 지연(100);

                } 부서지다;
        }
    }
```

24시간 알람 시계

이 응용 프로그램은 알람 기능이 있는 디지털 시계를 보여줍니다. Arduino의 전원이 켜지면 사용자가 시간을 설정할 때까지 다기능 쉴드 디스플레이가 깜박입니다. 시간 또는 알람을 설정하려면 버튼 1을 길게 누르십시오. 시간을 설정할 때 버튼 3을 사용하여 시간 또는 분을 설정합니다. 버튼 2를 누르십시오

알람 시간을 보거나 진행 중인 경우 알람을 취소합니다. 버튼 3을 누르고 있으면 알람이 활성화되거나 비활성화됩니다 (LED1은 알람이 활성화되었음을 나타냄). 이 응용 프로그램의 가능한 개선 사항은 다시 알림 기능을 사용하거나 하루 동안 장치를 여러 번 켜고 끄는 것입니다.

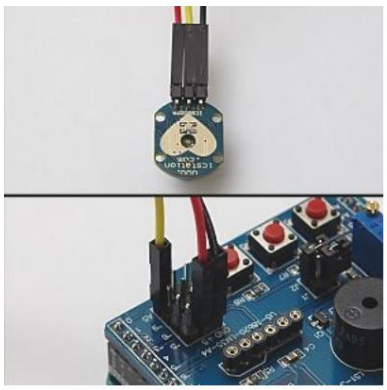
```
#include <MultiFuncShield.h>

/*
   버튼 1 버튼 2 버 : 누르고 있으면 시간 또는 알람이 설정됩니다. : 누르면
    튼 3 알람 시간을 보거나 진행 중인 경우 알람을 취소합니다. : (알람) 시간을 설정할 때 시/분 단위로 증가합니다. 토글하려면
    길게 누르세요.
알람 설정.

    LED1 : 켜짐 = 알람 활성화 */


휘발성 unsigned int clockMilliSeconds = 0; 휘발성 바이트 clockSeconds = 0; 휘발
성 바이트 clockMinutes = 0; 휘발성 바이트 clockHours = 12; 휘발성 바이트
clockEnabled = 1;



바이트 알람분 = 30; 바이트 알람 시간 = 6; 휘발
성 바이트 alarmEnabled = false;


바이트 alarmTogglePressed = 거짓;


열거형 displayModeValues {

    MODE_CLOCK_TIME,
    MODE_CLOCK_TIME_SET_HOUR,
    MODE_CLOCK_TIME_SET_MINUTE,
    MODE_ALARM_TIME,
    MODE_ALARM_TIME_SET_HOUR,
    MODE_ALARM_TIME_SET_MINUTE };


바이트 표시 모드 = MODE_CLOCK_TIME;

//---------------------------------------------------- ---------------------------------- 무효 설정() {


    MFS.userInterrupt = clockISR; MFS.initialize();


    MFS.blinkDisplay(DIGIT_ALL); //MFS.beep(500);

}


무효 루프() {

    // 여기에 기본 코드를 넣어 반복적으로 실행합니다.

    바이트 btn = MFS.getButton();

    스위치(디스플레이 모드) {

        케이스 MODE_CLOCK_TIME:
            displayTime(clockHours, clockMinutes);

            if (btn == BUTTON_2_PRESSED) {

                MFS.삐(0); // 알람을 취소합니다. 디스플레이 모드 =
                MODE_ALARM_TIME;

            } else if (btn == BUTTON_1_LONG_PRESSED) {

                MFS.blinkDisplay(DIGIT_ALL, OFF);
```

```
            MFS.blinkDisplay(DIGIT_1 | DIGIT_2); 디스플레이 모드 =
            MODE_CLOCK_TIME_SET_HOUR; 시계 활성화 = 거짓; clockMilliSeconds
            = 0; clockSeconds = 0;



        } else if (btn == BUTTON_3_LONG_PRESSED && !alarmTogglePressed) {

            alarmTogglePressed = 참; 알람 활성화 = !알람 활성
            화; MFS.writeLeds(LED_1, alarmEnabled);


        } else if (btn == BUTTON_3_LONG_RELEASE) {

            알람 토글 프레스 = 거짓;

        } 부서지다;

    케이스 MODE_CLOCK_TIME_SET_HOUR:
        if (btn == BUTTON_1_PRESSED) {

            MFS.blinkDisplay(DIGIT_1 | DIGIT_2, OFF); MFS.blinkDisplay(DIGIT_3 |
            DIGIT_4); 디스플레이 모드 = MODE_CLOCK_TIME_SET_MINUTE;


        } else if (btn == BUTTON_3_PRESSED || btn == BUTTON_3_LONG_PRESSED) {

            시계시간++; if (clockHours
            >= 24) {

                시계 시간 = 0;

            } displayTime(clockHours, clockMinutes);

        } 부서지다;

    경우 MODE_CLOCK_TIME_SET_MINUTE: if (btn ==
        BUTTON_1_PRESSED) {

            MFS.blinkDisplay(DIGIT_3 | DIGIT_4, OFF); 디스플레이 모드 =
            MODE_CLOCK_TIME; 시계 활성화 = 참;


        } else if (btn == BUTTON_3_PRESSED || btn == BUTTON_3_LONG_PRESSED) {

            시계분++; if (clockMinutes
            >= 60) {

                clockMinutes = 0;

            } displayTime(clockHours, clockMinutes);

        } 부서지다;

    케이스 MODE_ALARM_TIME:
        displayTime(alarmHours, alarmMinutes);

        if (btn == BUTTON_2_SHORT_RELEASE || btn == BUTTON_2_LONG_RELEASE) {

            디스플레이 모드 = MODE_CLOCK_TIME;

        } else if (btn == BUTTON_1_LONG_PRESSED) {

            MFS.blinkDisplay(DIGIT_ALL, OFF); MFS.blinkDisplay(DIGIT_1
            | DIGIT_2); 디스플레이 모드 = MODE_ALARM_TIME_SET_HOUR; 알람
            활성화 = 거짓;


        } 부서지다;

    경우 MODE_ALARM_TIME_SET_HOUR:
```

```
                if (btn == BUTTON_1_PRESSED) {

                    MFS.blinkDisplay(DIGIT_1 | DIGIT_2, OFF); MFS.blinkDisplay(DIGIT_3 |
                    DIGIT_4); 디스플레이 모드 = MODE_ALARM_TIME_SET_MINUTE;


                } else if (btn == BUTTON_3_PRESSED || btn == BUTTON_3_LONG_PRESSED) {

                    알람시간++; if
                    (alarmHours >= 24) {

                        알람 시간 = 0;

                    } displayTime(alarmHours, alarmMinutes);

                } 부서지다;

            경우 MODE_ALARM_TIME_SET_MINUTE: if (btn ==
                BUTTON_1_PRESSED) {

                    MFS.blinkDisplay(DIGIT_3 | DIGIT_4, OFF); 디스플레이 모드 =
                    MODE_CLOCK_TIME; 알람 활성화 = 참; MFS.writeLeds(LED_1, 켜짐);



                } else if (btn == BUTTON_3_PRESSED || btn == BUTTON_3_LONG_PRESSED) {

                    알람분++; if (alarmMinutes
                    >= 60) {

                        알람분 = 0;

                    } displayTime(alarmHours, alarmMinutes);

                } 부서지다;
        }
    }


    무효 displayTime (바이트 시간, 바이트 분) {

        문자 시간[5];

        sprintf(시간, "%03d", (시간 * 100) + 분); MFS.write(시간, 1);

    }

    //---------------------------------------------- ---------------------------------- 무효 clockISR() {


        // 모든 시간 구성 요소에 대해 리플 카운트를 수행합니다. if (clockEnabled) {


            clockMilliSeconds++; if
            (clockMilliSeconds >= 1000) {

                clockMilliSeconds = 0;

                시계초++; if (clockSeconds
                >= 60) {

                    clockSeconds = 0;

                    시계분++; if (clockMinutes
                    >= 60) {

                        clockMinutes = 0;

                        시계시간++; if (clockHours
                        >= 24) {
```

```
                    시계 시간 = 0;
                }
            }

            // 현재 시간이 알람 시간과 일치하고 알람이 활성화된 경우 작동
알람.
            if (alarmEnabled && (clockMinutes == alarmMinutes) && (clockHours ==
알람시간))
            {
                MFS.삐(
                                10, // 기간에
                                  5, // 오프 기간
                                  4, // 사이클 수
                               100, // 루프 사이클 수
                                50 // 루프 사이클 사이의 지연
                               );
            }
        }
    }
}
```

심장 모니터



이 응용 프로그램에는 LED의 빛을 사용하여 손가락이나 귓불의 맥박을 감지하는 이미지에 표시된 것과 유사한 심장 박동 센서(여러 인터넷 공급업체에서 구입할 수 있음)가 필요합니다. 정규화되면 센서가 아날로그 신호를 출력합니다.

심장 박동으로 인해 혈류가 바뀔 때마다 상승 및 하강하는 2.5볼트의 전압입니다. 애플리케이션은 신호가 3볼트 이상으로 상승한 다음 2.5볼트 아래로 떨어질 때마다 계산하여 작동합니다.
분당 비트를 계산하기 위해 펄스 사이의 시간을 측정합니다.
심장 박동이 감지될 때마다 신호음이 울립니다. 센서 출력은 다기능 실드에 노출된 Arduino 핀 A5에 연결해야 합니다.

경고: 이 응용 프로그램은 의료 목적을 위한 것이 아닙니다.

아두이노의 전원을 켠 후 검지 손가락을 센서에 부드럽게 그러나 단단히 대고 디스플레이가 깜박이기 시작할 때까지 기다립니다. 이것은 센서가 정상화되었음을 나타내며, 그 후에 실드 디스플레이에 분당 비트가 표시되고 신호음이 울립니다. 디스플레이가 0으로 유지되고 몇 초 후에도 깜박이지 않으면 손가락을 떼고 잠시 기다렸다가 다시 시도하십시오. 손가락이 일정한 압력으로 배치되었는지 확인하십시오.

이 응용 프로그램에 대한 가능한 수정은 각 심장 박동과 동기화된 LED 조명을 갖는 것입니다.

```
#include <MultiFuncShield.h>

무효 initializeSensorReading();
정수 데이터[4];
바이트 dataIdx=0;

바이트 펄스 감지 = 거짓;
정수 lastPulseTime = -1;


무효 설정() {
    // 여기에 설정 코드를 넣어 한 번 실행합니다.

    MFS.initialize();
    initializeSensorReading();
    //Serial.begin(9600);
```

```
}

무효 루프() {

    if (MFS.getTimer() == 0) {

        MFS.setTimer(10000); if (lastPulseTime !          // 밀리초 카운트다운 타이머를 재설정합니다.
        = -1) {

            lastPulseTime = 10000 + lastPulseTime;
        }
    }

    int sensorValue = analogRead(A5);                                    // 센서를 읽습니다.

    if (센서값 < 20 || 센서값 > 970) {

        // 센서가 정규화되지 않았습니다. 시간을 밀리초 단위로 확인합니다. if (lastPulseTime != -1 && lastPulseTime - MFS.getTimer() > 700) {


            initializeSensorReading();
        }

    } else if (sensorValue > (3 * 1024) / 5) // 값이 상승하므로 펄스의 시작이어야 합니다. {


        if (!pulseDetected) {

            펄스 감지 = 참;

            if (lastPulseTime == -1) {

                lastPulseTime = MFS.getTimer();

            } 또 다
            른 {
                int pulsePeriod = lastPulseTime - MFS.getTimer(); // 펄스 사이의 시간을 밀리초 단위로 계산합니다. lastPulseTime = MFS.getTimer();



                int bpm = 60000 / 펄스 주기;                                    // 분당 비트를 계산합니다.

                if (bpm < 45 || bpm > 230) 데이터 버퍼. {                    // bpm이 허용 범위를 벗어났으므로 명확합니다.


                    initializeSensorReading();

                } 또 다
                른 {
                    // bpm은 범위 내에 있지만 여전히 평균을 계산해야 합니다.

                    데이터[dataIdx++] = bpm;

                    if (dataIdx >= 4) {

                        데이터 ID = 0;
                    }

                    if (data[0] && data[1] && data[2] && data[3]) // 데이터 버퍼인지 확인
평균 bpm을 계산하기 전에 가득 찼습니다.
                    {
                        int avgBpm = (데이터[0] + 데이터[1] + 데이터[2] + 데이터[3]) / 4;

                        MFS.blinkDisplay(DIGIT_ALL, OFF); MFS.write(avgBpm);
                        MFS.beep();


                    } 또 다
                    른 {
                        // 버퍼가 가득 차지 않았으므로 디스플레이를 깜박입니다.
```

```
                    MFS.blinkDisplay(DIGIT_ALL, ON);
                }
            }
        }
    }
    else if (sensorValue < (1024 / 2)) // 값이 떨어지고 있으므로 펄스의 끝이어야 합니다.

    {
        펄스 감지 = 거짓;
    }

    //Serial.println(센서값);
    //지연(10);
}


// 읽기 버퍼를 초기화하고 표시합니다.
무효 initializeSensorReading()
{
    마지막 펄스 시간 = 0;

    데이터 ID = 0;
    for (int i=0; i<4; i++)
    {
        데이터[i] = 0;
    }

    MFS.write(0);
    MFS.blinkDisplay(DIGIT_ALL, OFF);
}
```

표면 경사 표시기

표면 경사 표시기 응용 프로그램은 MPU6050 모션 센서를 사용하여 각도를 결정합니다.
평평한 표면의 기울기. Arduino에 업로드하기 전에 전체 소스 코드를 다운로드해야 합니다. 이 응용 프로그램은 SDA 및 SCL에 각각 핀 A4 및 A5를 사용하여 Arduino Uno와 작동하도록 테스트되었습니다.

Arduino의 전원을 켠 후 가능한 한 평평한 표면에 모션 센서를 놓고 다기능 실드의 버튼 1을 길게 누릅니다. 센서가 보정되는 동안 LED 1이 깜박입니다. 그런 다음 경사면에 모션 센서를 배치하여 실드가 경사각을 표시합니다.

현재 경사는 단일 축에 대해서만 표시되지만 추가 축에 대한 경사를 표시하도록 애플리케이션을 수정할 수 있습니다.

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

#include "I2C.h"
#include "MPU6050.h"


무효 보정();

MPU6050 MPU;

const float radToDeg = 180.0 / 3.14159265358979323846264338327 95;
정수 x오프셋=0, y오프셋=0;
float zScaleOffset = 1; // Z축에 이 값을 곱하면 최대한 1g에 가까워집니다.
```

```
무효 설정() {
    // 여기에 설정 코드를 넣어 한 번 실행합니다.

    // 하드웨어 I2C 사용 Wire.begin();
    I2C1.initialize(&Wire);
    MPU.initialize(&I2C1,
    MPU_DEFAULT_ADDRESS, ACCEL_FS_2, GYRO_FS_250, DLPF_BW_5);

}

void loop() { // 여기에 메
    인 코드를 넣어 반복적으로 실행합니다.

    바이트 btn = MFS.getButton();

    if (btn == BUTTON_1_LONG_PRESSED) {

        보정();
    }

    MPU.getAccelRaw();

    MPU.accel_X_Raw -= xOffset; MPU.accel_Y_Raw
    -= yOffset;

    플로트 각도;

    if (MPU.accel_Z_Raw == 0) {

        각도 = 90;

    } 또 다른
    {
        각도 = atan((float)MPU.accel_Y_Raw / (MPU.accel_Z_Raw * zScaleOffset)) *
radToDeg; // y축 계산
        //각도 = atan((float)MPU.accel_X_Raw / (MPU.accel_Z_Raw * zScaleOffset)) * radToDeg; }
                        // X축 계산

    MFS.write(각도, 1); 지연(200);

}


무효 보정() {

    MFS.write(" ");
    MFS.writeLeds(LED_1, 켜짐);
    MFS.blinkLeds(LED_1, ON);

    // 처음 몇 개의 센서 판독값을 버립니다. for (int i=0; i<10; i++) {


        MPU.getAccelRaw(); 지연(10);

    }

    정수 x값[5], y값[5], z값[5];

    for (int i=0; i<5; i++) {

        MPU.getAccelRaw(); xValues[i]
        = MPU.accel_X_Raw; y값[i] = MPU.accel_Y_Raw;
        zValues[i] = MPU.accel_Z_Raw; 지연(300);



    } xOffset = MedianOf5(x값[0], x값[1], x값[2], x값[3], x값[4]); yOffset = MedianOf5(y값[0], y값[1], y값[2], y값[3], y값[4]);


    zScaleOffset = (float)MPU.accelScaleValue / MedianOf5(zValues[0], zValues[1],
z값[2], z값[3], z값[4]);
```

```
    MFS.blinkLeds(LED_1, OFF);

    // 입력 버튼 버퍼를 읽고 값을 삭제하여 삭제합니다. for (int i=0; i<10; i++) {


        MFS.getButton();
    }
}
```

소나 레인저

소나 레인저 애플리케이션은 HC SR04 소나 모듈을 사용하여 모듈과 최대 5미터 떨어진 고체 물체 사이의 거리를 측정합니다. 이 응용 프로그램은 주차 조작에서 운전자를 지원하는 일부 차량의 장애물 센서와 유사한 방식으로 작동합니다. 장애물이 소나 모듈에 가까워지면 신호음이 점점 더 짧은 간격으로 울립니다. 실드의 버튼 1은 소나 모듈을 결합하거나 분리하는 데 사용됩니다.

소나 모듈의 트리거 및 에코 핀은 다기능 실드에 노출된 Arduino 핀 5 및 6에 각각 연결됩니다. Arduino의 전원을 켠 후 소나 모듈에서 다른 거리에 단단한 물체를 놓습니다.

```
#include <MultiFuncShield.h>

const int TrigPin = 5; const int 에코핀 =
6;

enum sonarModeValues {

    MODE_SONAR_OFF,
    MODE_SONAR_ON };


바이트 소나 모드 = MODE_SONAR_OFF;

무효 설정() {

    //Serial.begin(9600); 핀모드(TrigPin,
    출력); 핀모드(에코핀, 입력);


    MFS.initialize(); MFS.write("꺼
    짐");
}

무효 루프() {

    바이트 btn = MFS.getButton();

    스위치(음파 탐지기 모드) {

        케이스 MODE_SONAR_OFF: if (btn
                == BUTTON_1_PRESSED) {

                    소나 모드 = MODE_SONAR_ON; MFS.삐(5, 95,
                    1,0,0); MFS.write("켜기");


                } 부서지다;

        케이스 MODE_SONAR_ON: if (btn
                == BUTTON_1_PRESSED) {

                    소나 모드 = MODE_SONAR_OFF; MFS.삐(0);
                    MFS.write("꺼짐");
```

```
                MFS.blinkDisplay(DIGIT_ALL, OFF); MFS.initSonar();


            } 또 다
            른 {
                int 거리 = MFS.getSonarDataCm(TrigPin, EchoPin);

                if (거리 != 0 && 거리 < 2000) {

                    int offPeriod = 거리 - 6;

                    if (offPeriod < 0) {

                        오프 기간 = 0;
                    }

                    MFS.write(거리);
                    MFS.setBeepOffPeriod(offPeriod);

                    MFS.blinkDisplay(DIGIT_ALL, 거리 < 11);

                } 지연(100);

            } 부서지다;
    }
}
```

속도계

속도계 응용 프로그램은 Arduino 핀 5에 연결된 자석과 리드 스위치를 사용하여 바퀴의 속도(킬로미터/시간)를 계산합니다. 또한 라인 또는 마크 센서를 사용하여 자신의 바퀴 인코더를 제작할 수 있어야 합니다.

Arduino의 전원을 켠 후 디스플레이가 깜박일 때까지 다기능 실드의 버튼 1을 누른 다음 버튼 2와 3을 사용하여 휠 직경을 센티미터로 설정합니다. 완료되면 버튼 1을 다시 누르십시오. 휠을 돌려 실드 디스플레이에 표시된 속도를 확인하십시오.

이 응용 프로그램의 가능한 개선 사항은 주행 거리를 킬로미터 단위로 기록하는 것입니다.

```
#include <MultiFuncShield.h>


열거형 SpeedoModeValues {

    SETUP_WHEEL,
    계산 속도 };


바이트 속도 모드 = CALCULATE_SPEED; 바이트 wheelDiameterCm
= 60; unsigned int wheelCirmcumferenceCm =
(wheelDiameterCm * 314) / 100;

float SpeedKmh(부호 없는 int wheelCircumferenceCm, 부호 없는 int periodMs);


무효 설정() {
    // 여기에 설정 코드를 넣어 한 번 실행합니다.

    핀모드(5, INPUT_PULLUP);

    MFS.initialize();

    MFS.initPulseInCounter( 5, // 펄스 입력을 위
            해 디지털 핀 5를 사용합니다. 2000, // 주기의 펄스를 0으로 재설정하기 전에 펄스를 기다리는 시간(밀리초)입니다.
```

```
            낮은                                    // LOW 입력에서 펄스를 트리거합니다.
    );
}


void loop() { // 여기에 메인
    코드를 넣어 반복적으로 실행합니다.

    바이트 btn = MFS.getButton();

    스위치(스피도모드) {

        케이스 SETUP_WHEEL: if (btn ==
                BUTTON_1_PRESSED) {

                    속도 모드 = CALCULATE_SPEED; MFS.blinkDisplay(DIGIT_ALL,
                    OFF); 휠 둘레 Cm = (휠 직경 Cm * 314) / 100;


                } else if (btn == BUTTON_2_PRESSED || btn == BUTTON_2_LONG_PRESSED) {

                    wheelDiameterCm--;

                    if (wheelDiameterCm < 30) {

                        휠 직경Cm = 30;
                    }
                    MFS.write(wheelDiameterCm);

                } else if (btn == BUTTON_3_PRESSED || btn == BUTTON_3_LONG_PRESSED) {

                    휠직경Cm++;

                    if (wheelDiameterCm > 100) {

                        휠 직경Cm = 100;
                    }
                    MFS.write(wheelDiameterCm);

                } 부서지다;

        경우 CALCULATE_SPEED: if (btn ==
                BUTTON_1_LONG_PRESSED) {

                    속도 모드 = SETUP_WHEEL;
                    MFS.write(wheelDiameterCm);
                    MFS.blinkDisplay(DIGIT_ALL, ON);

                } 또 다
                른 {
                    부호 없는 정수 pulsePeriodMs = MFS.getPulseInPeriod();

                    if (펄스 주기 == 0) {

                        MFS.write(0.0, 1);

                    } 또 다른
                    {
                        MFS.write(SpeedKmh(wheelCirmcumferenceCm, pulsePeriodMs), 1);
                    }

                } 부서지다;

    } 지연(100);
}


float SpeedKmh(부호 없는 int wheelCircumferenceCm, 부호 없는 int periodMs) {

    return (float)(wheelCircumferenceCm * 36) / periodMs;
}
```

모든 코드 샘플과 애플리케이션은 테스트를 거쳤으며 작동합니다. 어려움을 겪으셨다면,
의견을 게시해 주시면 최대한 빨리 연락 드리겠습니다.

부록

다기능 실드 라이브러리 도움말

```
#정의 ON 1
#define OFF 0

#define LED_1_PIN #define 13
LED_2_PIN #define LED_3_PIN 12
#define LED_4_PIN #define 11
POT_PIN #define BEEPER_PIN 10
#define BUTTON_1_PIN A1 0
삼

#define BUTTON_2_PIN A2
#define BUTTON_3_PIN A3
#define LATCH_PIN #define 4
CLK_PIN #define DATA_PIN 7
#define LM35_PIN 8
A4

#DIGIT_1 정의 1
#define DIGIT_2 2
#define DIGIT_3 4
#define DIGIT_4 8
#DIGIT_ALL 15 정의

#define LED_1 1
#define LED_2 2
#define LED_3 4
#define LED_4 8
#define LED_ALL 15

// 버튼 상태 표시기
#define BUTTON_PRESSED_IND(0 << 6)
#define BUTTON_SHORT_RELEASE_IND(1 << 6)
#define BUTTON_LONG_PRESSED_IND(2 << 6)
#define BUTTON_LONG_RELEASE_IND(3 << 6)

#define BUTTON_1_PRESSED #define (1 | BUTTON_PRESSED_IND)
BUTTON_1_SHORT_RELEASE(1 | BUTTON_SHORT_RELEASE_IND)
#define BUTTON_1_LONG_PRESSED(1 | BUTTON_LONG_PRESSED_IND)
#define BUTTON_1_LONG_RELEASE(1 | BUTTON_LONG_RELEASE_IND)

#define BUTTON_2_PRESSED #define (2 | BUTTON_PRESSED_IND)
BUTTON_2_SHORT_RELEASE(2 | BUTTON_SHORT_RELEASE_IND)
#define BUTTON_2_LONG_PRESSED(2 | BUTTON_LONG_PRESSED_IND)
#define BUTTON_2_LONG_RELEASE(2 | BUTTON_LONG_RELEASE_IND)

#define BUTTON_3_PRESSED #define (3 | BUTTON_PRESSED_IND)
BUTTON_3_SHORT_RELEASE(3 | BUTTON_SHORT_RELEASE_IND)
#define BUTTON_3_LONG_PRESSED(3 | BUTTON_LONG_PRESSED_IND)
#define BUTTON_3_LONG_RELEASE(3 | BUTTON_LONG_RELEASE_IND)

#define BUTTON_COUNT #define 삼
SMOOTHING_NONE #define 0
SMOOTHING_MODERATE #define 1
SMOOTHING_STRONG 2


클래스 MultiFuncShield
{

    공공의:
        // 1khz의 주파수로 사용자 인터럽트에 대한 포인터.
        무효(*userInterrupt)() = NULL;
```

```
// 이 인스턴스를 초기화하지만 인터럽트 기반 기능을 사용할 수 없습니다.
무효 초기화();

// 내부용으로만 사용합니다.
무효 isrCallBack();

// 밀리초 카운트다운 타이머를 시작합니다.
void setTimer(부호 없는 1/1000 단위);

// 카운트다운 타이머의 현재 값을 가져옵니다.
서명되지 않은 긴 getTimer();

// 밀리초 카운트다운 타이머를 시작하고 0에 도달할 때까지 기다립니다.
무효 대기(부호 없는 1/1000 단위);

// LED 숫자 디스플레이에 씁니다.
무효 쓰기(const char *텍스트 문자열, 바이트 rightJustify =0);
무효 쓰기(정수 정수);
무효 쓰기(부동수, 바이트 decimalPlaces = 1);

// Led 숫자 디스플레이를 수동으로 새로 고칩니다.
// 인터럽트 기반 기능을 사용할 수 있는 동안에는 사용되지 않습니다.
무효 manualDisplayRefresh();

// LED 숫자 디스플레이의 숫자를 깜박입니다.
void 깜박임 디스플레이(바이트 숫자, 바이트 활성화 =
                                        ON);          // 비트 또는 예를 들어 DIGIT_1 | DIGIT_2
                                                      // 깜박임 켜기/끄기

무효 setDisplayBrightness(바이트 수준); // 0 = 최대, 3 = 최소

// LED를 켜거나 끕니다.
무효 writeLeds(바이트 leds, 바이트 조명);          // 비트 또는 예를 들어 LED_1 사용 | LED_2
                                                      // 켜짐 또는 꺼짐

// LED를 깜박입니다.
void 깜박임Leds(바이트 LED, 바이트 활성화 =
                                        ON );         // 비트 또는 예를 들어 LED_1 사용 | LED_2
                                                      // 켜짐 또는 꺼짐

// 백그라운드에서 관리되는 신호음을 작동시킵니다. 기간 타이밍은 100번째입니다.
초의
무효 beep(unsigned int onPeriod = 20, unsigned int offPeriod = 0, 바이트 주기 = 1,
unsigned int loopCycles = 1 /* 0=무한 */, unsigned int loopDelayPeriod =0);

// 신호음이 작동하는 동안 꺼짐 기간을 설정하려면 이것을 사용합니다.
무효 setBeepOffPeriod(부호 없는 int offPeriod);

// 버튼 작업을 버튼 대기열에 추가합니다(예: BUTTON_1_PRESSED).
무효 queueButton(바이트 버튼);

// 버튼 큐에서 버튼 액션을 가져옵니다.
바이트 getButton();

// 버튼을 짧게 누르고 떼는 작업을 대기열에 추가합니다. 버튼을 길게 누르는 것은
지원되며 긴 릴리스는 짧은 릴리스로 보고됩니다.
// 인터럽트 기반 기능을 사용할 수 있는 동안 사용하면 안 됩니다.
무효 manualButtonHandler();

// 펄스 카운터를 초기화합니다. 입력 핀에 인가되는 펄스를 카운팅할 때 사용합니다.
최대 펄스 주파수 500hz.
무효 initPulseInCounter(바이트 핀 = BUTTON_1_PIN, 부호 없는 int timeOut = 3000,   // 입력 핀
                                                                                    // 밀리초 수
주기의 펄스를 0으로 재설정하기 전에 펄스를 기다립니다.
                                                바이트 트리거 = 낮음                // 둘 중 하나에서 트리거 카운터
상승 또는 하강 에지
                                                );

무효 펄스 인카운터();
```

```
        // 가장 최근 펄스의 주기를 가져옵니다(밀리초).
        부호 없는 int getPulseInPeriod();

        // 카운트된 총 펄스 수를 가져옵니다.
        서명되지 않은 긴 getPulseInTotalCount();

        // 펄스 카운터를 0으로 재설정합니다.
        무효 resetPulseInTotalCount();

        // 펄스를 시간 초과로 설정합니다. 이는 대기 시간을 밀리초 단위로 나타냅니다.
펄스, 주기의 펄스를 0으로 재설정하기 전에.
        무효 setPulseInTimeOut(부호 없는 int timeOut);

        // 소나 판독 기능을 초기화합니다. HC-SR04 소나 모듈이 필요합니다.
        무효 initSonar(바이트 레벨 = SMOOTHING_MODERATE); // 레벨 0=없음, 1=보통, 2=강함.


        // HC-SR04 소나 모듈을 사용하여 센티미터 단위로 측정된 거리를 가져옵니다.
        unsigned int getSonarDataCm(바이트 triggerPin, 바이트 echoPin);

        // 온도 읽기 기능을 초기화합니다. LM35 센서가 필요합니다. 점퍼 J1을 제거해야 합니다.
방패에서.
        무효 initLM35(바이트 수준 = SMOOTHING_MODERATE); // 레벨 0=없음, 1=보통, 2=강함

        // 섭씨 1/10 단위로 온도 판독값을 가져옵니다.
        int getLM35Data();

    사적인:
        TimerOne *timer1;
        휘발성 바이트 timerReadInProgress = 0;
        휘발성 바이트 timerWriteInProgress = 0;

        //const 바이트 buttonPins[3] = {BUTTON_1_PIN, BUTTON_2_PIN, BUTTON_3_PIN}; // 위의 버튼 매크로와 일치해야 합니다.


        휘발성 바이트 buttonBuffer[BUTTON_COUNT * 2];
        휘발성 char buttonBufferCount = 0;
        휘발성 바이트 button_write_pos = 0;
        휘발성 바이트 button_read_pos = 0;

        서명되지 않은 int buttonSampleIntervalCounter = 0;
        바이트 버튼 상태[BUTTON_COUNT] = {0,0,0}; // 현재 업 또는 다운 상태
        unsigned int buttonPressTime[BUTTON_COUNT] = {0,0,0};

        volatile unsigned long timer_volatile = 0;                                  // 1000분의 1초 카운트다운 타이머
해결.
        휘발성 서명되지 않은 긴 timer_safe = 0;

        휘발성 바이트 beeperModifyInProgress = 0;
        휘발성 바이트 beeperState = 0; // 0=온 기간; 1=오프 기간
        휘발성 unsigned int beeperOnPeriodReloadValue = 0;
        휘발성 unsigned int beeperOffPeriodReloadValue = 0;
        휘발성 unsigned int beeperPeriodCounter = 0;
        휘발성 바이트 beeperCycleReloadValue = 0;
        휘발성 바이트 beeperCycleCounter = 0;
        휘발성 unsigned int beeperLoopCycleCounter = 0;
        휘발성 unsigned int beeperLoopDelayPeriodReloadValue = 0;

        바이트 displayIdx = 0;
        바이트 깜박임 활성화 = 0; // 표시 숫자에 매핑된 최하위 비트.
        바이트 깜박임 상태 = 0;
        바이트 깜박임 카운터 = 0;

        //바이트 ledControlMask = 0; TImerOne 라                    // 소프트 활성화/비활성화 LED. 다음과 같은 경우 여기에서 LED를 비활성화합니다.
이브러리의 PWM을 사용합니다. 바이트 ledState = 0; 바이
        트 ledBlinkEnabled = 0; 바이트 ledOutput=0;          // LED에 매핑된 최하위 비트
        깜박거리다)                                          // LED에 매핑된 최하위 비트
                                                             // 현재 led 출력(고려


        휘발성 바이트 pulseInEnabled = false;
```

```
        휘발성 바이트 pulseInReadInProgress = 0; 휘발성 바이트
        pulseInWriteInProgress = 0;

        휘발성 부호 없는 정수 pulseInTimeOut = 3000; // 펄스 주기를 측정하기 위한 시간 프레임. 휘발성 바이트 pulseInPin = BUTTON_1_PIN; 휘발성 unsigned
int pulseInPeriodCounter = 3000; 휘발성 바이트 pulseInTrigger = 낮음; // LOW 또는 HIGH 휘발성에서 트리거합니다. unsigned int pulseInPeriod_volatile =0;
        휘발성 unsigned int pulseInPeriod_safe = 0; 휘발성 바이트 pulseInState = 0; 휘발성 부호 없는 긴 pulseInTotalCount_volatile = 0; 휘발성 부호 없는 긴
        pulseInTotalCount_safe = 0;









        바이트 소나스무딩레벨 = SMOOTHING_MODERATE; 바이트 lm35SmoothingLevel =
        SMOOTHING_MODERATE;
};

extern MultiFuncShield MFS;
```

MPU6050 도움말

클래스 MPU6050 {

공공의:

int accel_X_Raw; int
accel_Y_Raw; 정수
accel_Z_Raw;

int 자이로_X_Raw; int
gyro_Y_Raw; int 자이로
_Z_Raw;

정수 가속도 값; // 원시 가속을 이 값으로 나누어 g 단위로 읽습니다. 부동 자이로 스케일 값; // 원시 자이로를 이 값으로 나누어 도/초 회전 속도를 얻습니다.

// MPU6050 센서를 초기화합니다. 무효 초기화(II2C *i2c, 바이트 가
산기, 바이트 accelScale = 0, 바이트 gyroScale = 0, 바이트 dlpf = 0);

무효 getAccelRaw(); // 원시 가속 값을 가져옵니다. void getGyroRaw(); // 자이로 가속도 값 얻기
int getTemp10th(); // 섭씨 10분의 1도의 온도를 얻습니다. };

#define ADDRESS_AD0_LOW 0x68 #define
ADDRESS_AD0_HIGH 0x69 #define
MPU_DEFAULT_ADDRESS ADDRESS_AD0_LOW

// 가속 스케일 #define
ACCEL_FS_2 0x00 #define ACCEL_FS_4
0x01 #define ACCEL_FS_8 0x02 #define
ACCEL_FS_16 0x03

// 자이로 스케일 #define
GYRO_FS_250 0x00 #define GYRO_FS_500
0x01 #define GYRO_FS_1000 0x02 #define
GYRO_FS_2000 0x03

// dlpf
#define DLPF_BW_256 0x00 #define
DLPF_BW_188 0x01 #define DLPF_BW_98
0x02 #define DLPF_BW_42 0x03 #define
DLPF_BW_0x0x04 #define_5DL