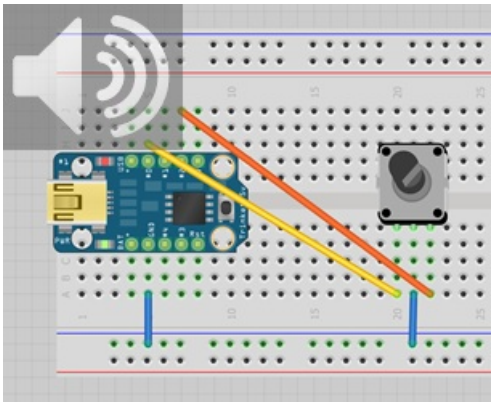
adafruit learning system

Trinket USB Volume Knob

Created by Frank Zhao



Last updated on 2020-01-23 04:58:56 PM UTC

Overview

One thing I miss from my old stereo system is the big volume knob, maybe you wish your computer or laptop had a volume knob too, too? This tutorial will show how you can build a little desktop toy that lets you quickly set the volume without having to open up iTunes or your control panel or run **amixer**

If you have a rotary encoder with a built-in switch, you can even extend this project to turn it into a volume/mute control

How it works!

The Trinket's USB port is used for uploading sketches, but you can **also** use it for some basic USB 1.1 devices. For example, under USB 1.1, you can make low speed USB devices such as...

- HID devices
    - mouse
    - keyboard
    - joystick/gamepad/flightsim
    - many more... the drivers come from the operating system
- MIDI devices (input notes from sensors, or generate outputs from notes, etc)
- Custom devices (you have to write your own driver though)
- There are a few more rare applications

However, USB 1.1 cannot do stuff like virtual serial ports or mass storage devices, these require USB 2.0, sorry.

The easiest and most useful are HID devices. Such as a USB keyboard (link to another tutorial) (https://adafru.it/cNB).

Today, keyboards have multimedia keys, such as play/pause or volume control. I will show you how to combine a Trinket and a rotary encoder to create a USB volume control knob.

Wiring

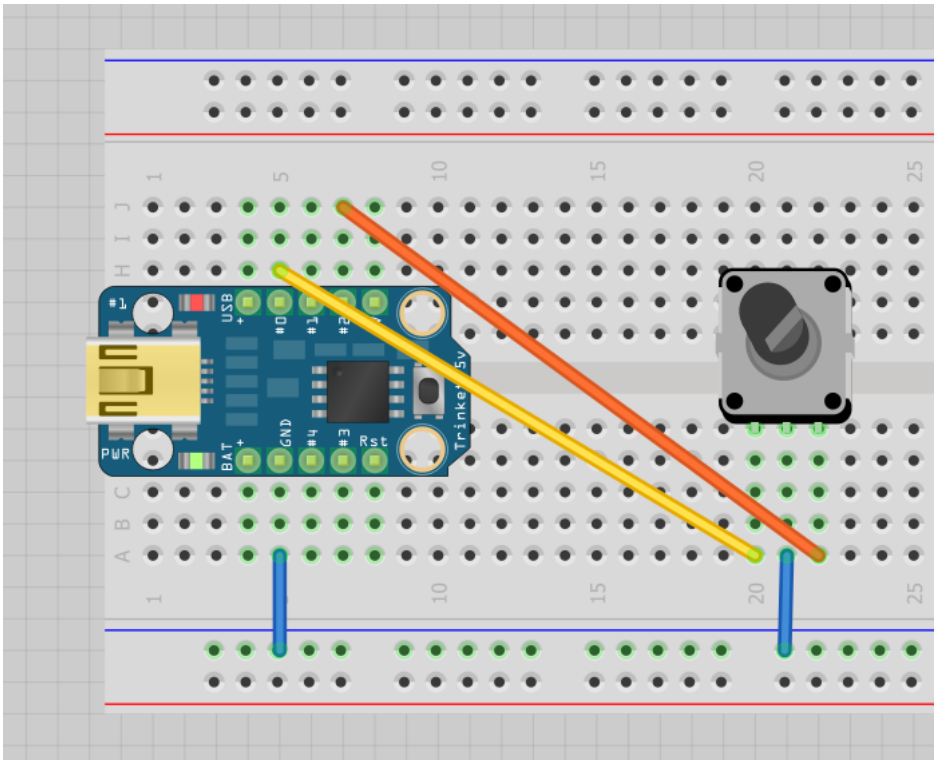
Connect Trinket's ground to the rotary encoder's common pin.

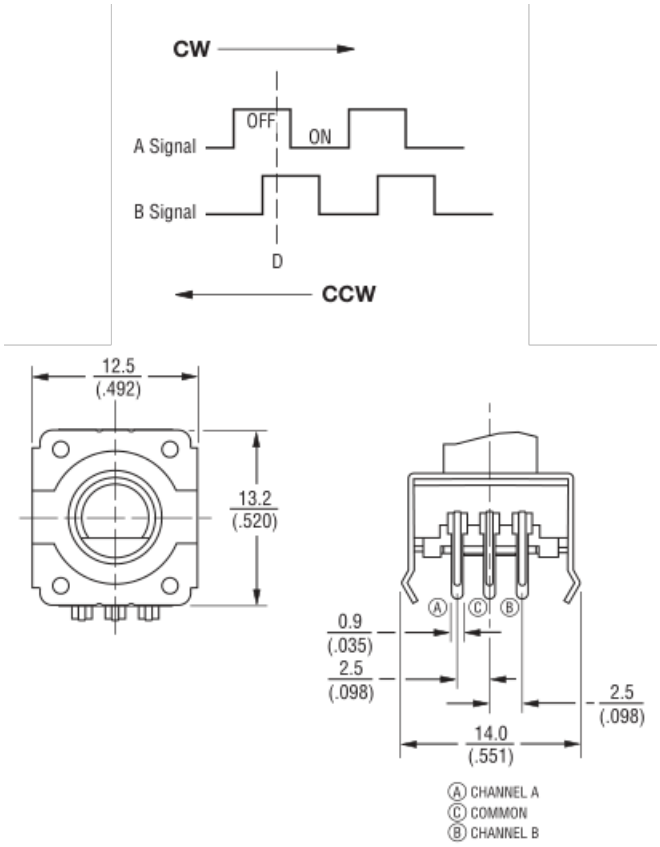
Connect Trinket's pin #0 to the rotary encoder's "A" signal pin.

Connect Trinket's pin #2 to the rotary encoder's "B" signal pin.

It's that simple! If you want, you can put it into a little box, or make a desk stand for it, or mount it on your PC tower, be creative.

These encoder signals will be "active-low". Each of these signals is a switch inside the rotary encoder. Active-low means the other end of the switch is connected to ground, such that when the switch is "closed", the pin value will read low. We will be using the Trinket's internal pull-up resistors so when the switch is "open", the pin value will read high.



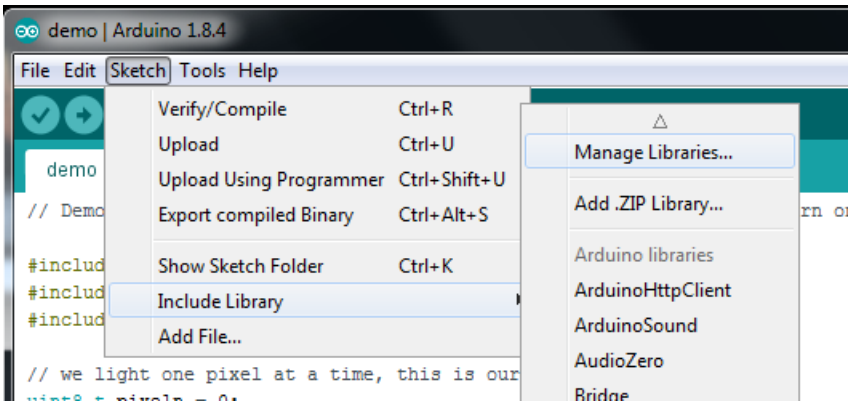

CW
OFF
ON
A Signal
B Signal
D
CCW
12.5
(.492)
13.2
(.520)
0.9
(.035)
2.5
(.098)
2.5
(.098)
14.0
(.551)
Ⓐ CHANNEL A
Ⓒ COMMON
Ⓑ CHANNEL B

Code

The libraries are located at our github repository https://github.com/adafruit/Adafruit-Trinket-USB/ (https://adafru.it/cJL) , this particular project is the example project under TrinketHidCombo, named TrinketVolumeKnob.

Install these as ordinary Arduino libraries, via the Arduino Library Manager (see http://arduino.cc/en/Guide/Libraries (https://adafru.it/cJM) or our tutorial on Arduino libraries (https://adafru.it/aYM)). Keep in mind that this library is designed specifically for Trinket. (although they can be modified to work with other platforms).

Open up the Arduino Library Manager:



Search for the **Trinket USB Keyboard** library and install it

The code exhibits a basic implementation of a polling rotary encoder reading technique.

Pins are first setup, and USB library is initialized. If a rotation is detected, then the USB library sends out a keystroke.

```
#include "TrinketHidCombo.h"

#define PIN_ENCODER_A 0
#define PIN_ENCODER_B 2
#define TRINKET_PINx  PINB

static uint8_t enc_prev_pos = 0;
static uint8_t enc_flags    = 0;

void setup()
{
  // set pins as input with internal pull-up resistors enabled
  pinMode(PIN_ENCODER_A, INPUT);
  pinMode(PIN_ENCODER_B, INPUT);
  digitalWrite(PIN_ENCODER_A, HIGH);
  digitalWrite(PIN_ENCODER_B, HIGH);

  TrinketHidCombo.begin(); // start the USB device engine and enumerate
```

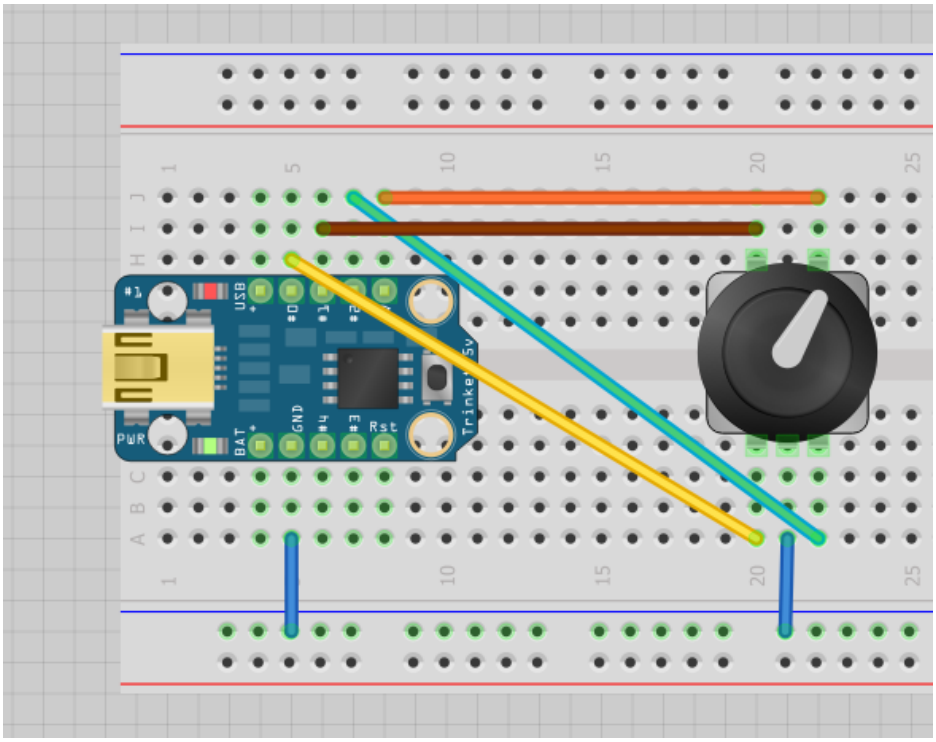
```
  // get an initial reading on the encoder pins
  if (digitalRead(PIN_ENCODER_A) == LOW) {
    enc_prev_pos |= (1 << 0);
  }
  if (digitalRead(PIN_ENCODER_B) == LOW) {
    enc_prev_pos |= (1 << 1);
  }
}

void loop()
{
  int8_t enc_action = 0; // 1 or -1 if moved, sign is direction

  // note: for better performance, the code will now use
  // direct port access techniques
  // http://www.arduino.cc/en/Reference/PortManipulation
  uint8_t enc_cur_pos = 0;
  // read in the encoder state first
  if (bit_is_clear(TRINKET_PINx, PIN_ENCODER_A)) {
    enc_cur_pos |= (1 << 0);
  }
  if (bit_is_clear(TRINKET_PINx, PIN_ENCODER_B)) {
    enc_cur_pos |= (1 << 1);
  }

  // if any rotation at all
  if (enc_cur_pos != enc_prev_pos)
  {
    if (enc_prev_pos == 0x00)
    {
      // this is the first edge
      if (enc_cur_pos == 0x01) {
        enc_flags |= (1 << 0);
      }
      else if (enc_cur_pos == 0x02) {
        enc_flags |= (1 << 1);
      }
    }

    if (enc_cur_pos == 0x03)
    {
      // this is when the encoder is in the middle of a "step"
      enc_flags |= (1 << 4);
    }
    else if (enc_cur_pos == 0x00)
    {
      // this is the final edge
      if (enc_prev_pos == 0x02) {
        enc_flags |= (1 << 2);
      }
      else if (enc_prev_pos == 0x01) {
        enc_flags |= (1 << 3);
      }

      // check the first and last edge
      // or maybe one edge is missing, if missing then require the middle state
      // this will reject bounces and false movements
      if (bit_is_set(enc_flags, 0) && (bit_is_set(enc_flags, 2) || bit_is_set(enc_flags, 4))) {
        enc_action = 1;
      }
```

```
      }
      else if (bit_is_set(enc_flags, 2) && (bit_is_set(enc_flags, 0) || bit_is_set(enc_flags, 4))) {
        enc_action = 1;
      }
      else if (bit_is_set(enc_flags, 1) && (bit_is_set(enc_flags, 3) || bit_is_set(enc_flags, 4))) {
        enc_action = -1;
      }
      else if (bit_is_set(enc_flags, 3) && (bit_is_set(enc_flags, 1) || bit_is_set(enc_flags, 4))) {
        enc_action = -1;
      }

      enc_flags = 0; // reset for next time
    }
  }

  enc_prev_pos = enc_cur_pos;

  if (enc_action > 0) {
    TrinketHidCombo.pressMultimediaKey(MMKEY_VOL_UP);
  }
  else if (enc_action < 0) {
    TrinketHidCombo.pressMultimediaKey(MMKEY_VOL_DOWN);
  }
  else {
    TrinketHidCombo.poll(); // do nothing, check if USB needs anything done
  }
}
```

Add a Mute Button

The rotary encoder sold in the Adafruit store has a shaft that is also a button! To extend on the first example, the second example sketch will use this feature to add a mute function.

See the new wiring diagram, which has two more wires. The button switch is connected to Trinket's pin #1. It is "active-high", meaning when the button is pressed, the pin will read as logic high. When the button is not pressed, the pin will read as logic low. (the built-in LED of the Trinket shares pin #1, the LED acts as a pull-down resistor)



The newer code is the same old code but adds the mute button function.

```
#include "TrinketHidCombo.h"

#define PIN_ENCODER_A 0
#define PIN_ENCODER_B 2
#define TRINKET_PINx  PINB
#define PIN_ENCODER_SWITCH 1

static uint8_t enc_prev_pos   = 0;
static uint8_t enc_flags      = 0;
static char    sw_was_pressed = 0;

void setup()
{
  // set pins as input with internal pull-up resistors enabled
  pinMode(PIN_ENCODER_A, INPUT);
  pinMode(PIN_ENCODER_B, INPUT);
  digitalWrite(PIN_ENCODER_A, HIGH);
  digitalWrite(PIN_ENCODER_B, HIGH);
```

```
  pinMode(PIN_ENCODER_SWITCH, INPUT);
  // the switch is active-high, not active-low
  // since it shares the pin with Trinket's built-in LED
  // the LED acts as a pull-down resistor
  digitalWrite(PIN_ENCODER_SWITCH, LOW);

  TrinketHidCombo.begin(); // start the USB device engine and enumerate

  // get an initial reading on the encoder pins
  if (digitalRead(PIN_ENCODER_A) == LOW) {
    enc_prev_pos |= (1 << 0);
  }
  if (digitalRead(PIN_ENCODER_B) == LOW) {
    enc_prev_pos |= (1 << 1);
  }
}

void loop()
{
  int8_t enc_action = 0; // 1 or -1 if moved, sign is direction

  // note: for better performance, the code will now use
  // direct port access techniques
  // http://www.arduino.cc/en/Reference/PortManipulation
  uint8_t enc_cur_pos = 0;
  // read in the encoder state first
  if (bit_is_clear(TRINKET_PINx, PIN_ENCODER_A)) {
    enc_cur_pos |= (1 << 0);
  }
  if (bit_is_clear(TRINKET_PINx, PIN_ENCODER_B)) {
    enc_cur_pos |= (1 << 1);
  }

  // if any rotation at all
  if (enc_cur_pos != enc_prev_pos)
  {
    if (enc_prev_pos == 0x00)
    {
      // this is the first edge
      if (enc_cur_pos == 0x01) {
        enc_flags |= (1 << 0);
      }
      else if (enc_cur_pos == 0x02) {
        enc_flags |= (1 << 1);
      }
    }

    if (enc_cur_pos == 0x03)
    {
      // this is when the encoder is in the middle of a "step"
      enc_flags |= (1 << 4);
    }
    else if (enc_cur_pos == 0x00)
    {
      // this is the final edge
      if (enc_prev_pos == 0x02) {
        enc_flags |= (1 << 2);
      }
      else if (enc_prev_pos == 0x01) {
        enc_flags |= (1 << 3);
```

```
      enc_flags |= (1 << 3);
    }

    // check the first and last edge
    // or maybe one edge is missing, if missing then require the middle state
    // this will reject bounces and false movements
    if (bit_is_set(enc_flags, 0) && (bit_is_set(enc_flags, 2) || bit_is_set(enc_flags, 4))) {
      enc_action = 1;
    }
    else if (bit_is_set(enc_flags, 2) && (bit_is_set(enc_flags, 0) || bit_is_set(enc_flags, 4))) {
      enc_action = 1;
    }
    else if (bit_is_set(enc_flags, 1) && (bit_is_set(enc_flags, 3) || bit_is_set(enc_flags, 4))) {
      enc_action = -1;
    }
    else if (bit_is_set(enc_flags, 3) && (bit_is_set(enc_flags, 1) || bit_is_set(enc_flags, 4))) {
      enc_action = -1;
    }

    enc_flags = 0; // reset for next time
  }
 }

 enc_prev_pos = enc_cur_pos;

 if (enc_action > 0) {
   TrinketHidCombo.pressMultimediaKey(MMKEY_VOL_UP);
 }
 else if (enc_action < 0) {
   TrinketHidCombo.pressMultimediaKey(MMKEY_VOL_DOWN);
 }

 // remember that the switch is active-high
 if (bit_is_set(TRINKET_PINx, PIN_ENCODER_SWITCH))
 {
   if (sw_was_pressed == 0) // only on initial press, so the keystroke is not repeated while the
button is held down
   {
     TrinketHidCombo.pressMultimediaKey(MMKEY_MUTE);
     delay(5); // debounce delay
   }
   sw_was_pressed = 1;
 }
 else
 {
   if (sw_was_pressed != 0) {
     delay(5); // debounce delay
   }
   sw_was_pressed = 0;
 }

 TrinketHidCombo.poll(); // check if USB needs anything done
}
```

Learn More

Trinket is based on V-USB (https://adafru.it/cJO). It is a bit-bang implementation of USB. The generation of signals is done through assembly code, outputting 1s and 0s to the USB signal pins with precise timing.

V-USB is designed for AVR microcontrollers without native USB capabilities. USB signals have tight timing specifications that must be met, which is why the timing critical portions of V-USB are written in assembly code. Assembly instructions have predictable execution times and thus it is easier to calculate timing.

This is what makes V-USB so cool! Most of the microcontrollers on the market with native USB are big, but V-USB can turn a 8 pin ATtiny85 into an USB device!

To learn more about V-USB, check out the example projects on the V-USB website:
http://www.obdev.at/products/vusb/projects.html (https://adafru.it/cJP)

Trinket can do almost any of those projects, as long as there are enough pins and enough flash memory to use.

Beware however, like I've mentioned in the overview, V-USB is only capable of creating low speed USB 1.1 devices. If you need to create USB 2.0 devices, you need a microcontroller with native USB such as the ATmega32u4 (used in the Arduino Micro & Leonardo), not "bit-banged" USB.