USB SNES Gamepad

Created by lady ada

https://learn.adafruit.com/usb-snes-gamepad

Last updated on 2022-12-01 01:48:46 PM EST

Table of Contents

Overview

This project tutorial will show you how you can convert a console game pad into a USB keyboard mouse for playing games on your PC. The USB game pad can be used with nearly any software, such as a MAME emulator, game, simulation software, or for custom user interfaces.

We'll start by turning the buttons of the game pad into keyboard buttons, so that pressing 'up' is converted into the 'U' key, for example. The firmware is easily adaptable, so you can adjust it for whatever software it will be used with.

Then we'll make the project more interesting by adding an accelerometer. This will allow the game pad to be used as a mouse by tilting it!

This tutorial including the original code and Portal video is by Devlin Thyne ()! Rock!

What you'll need:

You'll need the following in order to build the project:

- Game Pad Controller (http://adafru.it/131) - We'll be using an SNES Controller
- Teensy (http://adafru.it/199) - This is a very small microcontroller board that can act as a keyboard/mouse
- Triple-axis accelerometer (http://adafru.it/163) - We'll be using the nice ADXL335 on a breakout board. You can skip this if you're not planning to add in the mouse capability
- USB cable with mini-b connector (http://adafru.it/260) - to attach to the Teensy for plugging into a computer!
- Ribbon cable - for all the soldering connections. Rainbow cable is the easiest to work with as its color coded

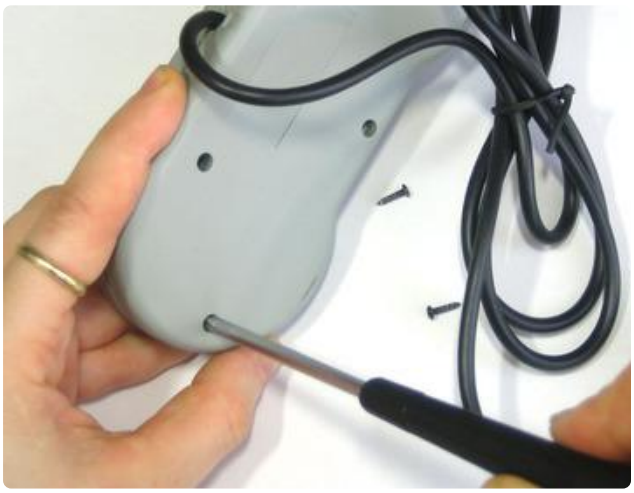
If you want to build the entire project, we have a project pack in the shop with all the parts listed above! (http://adafru.it/241)

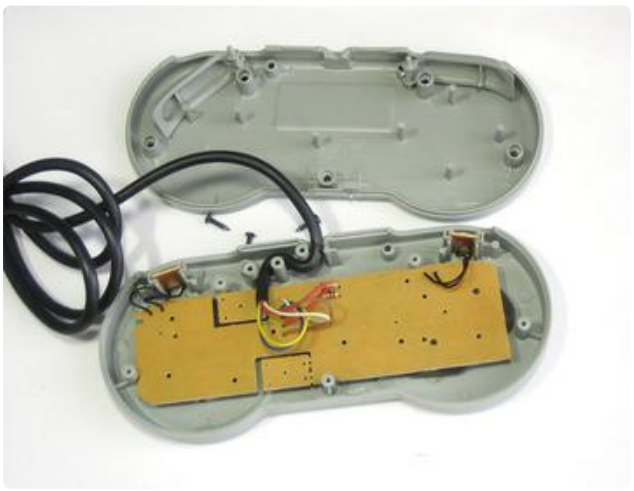
You'll also need some basic hand tools such as screwdrivers, wire strippers, soldering iron (http://adafru.it/180), solder (http://adafru.it/145), diagonal cutters (http://adafru.it/152), vise or third hand tool, etc.

All the code is on GitHub, including some extra sketches we've written () so be sure to look there!

Disassemble the SNES Controller

We'll begin by disassembling the SNES controller.





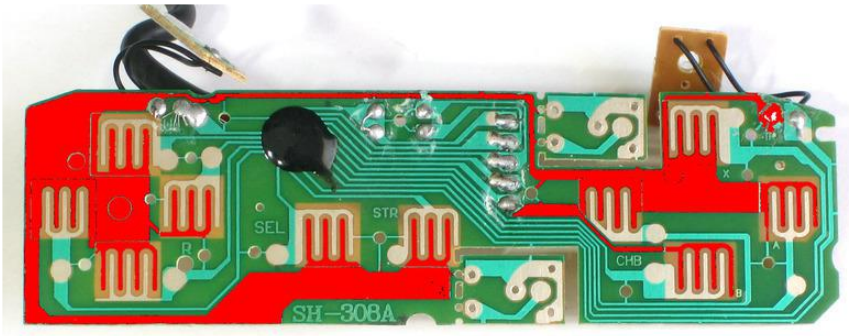
There are 5 small phillips screws on the back. Once you lift the back off, you can remove the PCB. Be careful as there are tiny wires for the 'side' buttons so just make sure those pieces come out cleanly.

Each button is made of 3 parts - theres the plastic part that you press, beneath that is the elastomer which is a rubber molded piece with a conductive bit that goes underneath the plastic part, and finally on the PCB there are two interdigitated and exposed traces. When the user presses the plastic button, it pushes down on the elastomer which then pushes the conductive rubber onto both traces, shorting them.

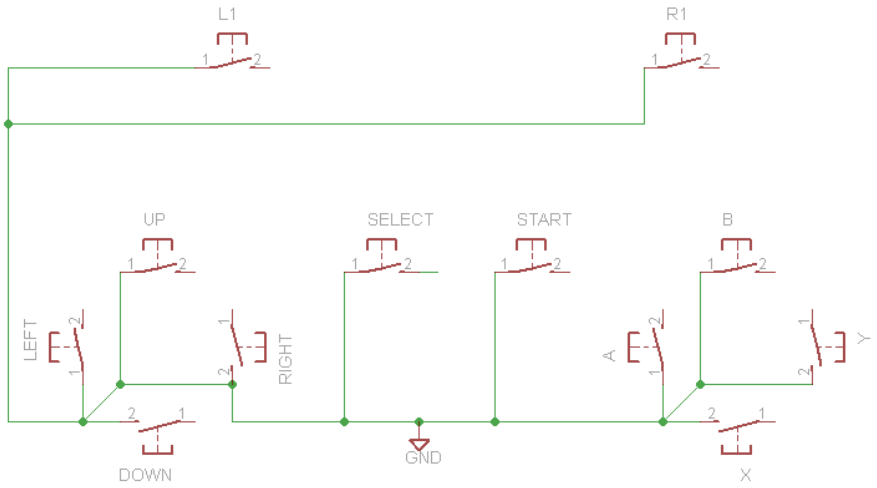
There is also a black blob in the middle. This blob is a chip that takes all the button inputs and then converts it into the way that the SNES wants to hear. Thats all fine, but we dont want to use the blob because we are going to make our own custom chip software. (Note that it would be pretty easy to make the Teensy 'talk' right to the blob using the SNES protocol but then you wouldn't be able to adapt this tutorial to other controllers, for that reason we're going to do it the 'hard way')

The question is now how can we listen to all the buttons?

Well, luckily, almost all game pads are going to use a similar method for arranging the buttons. If you note carefully at the PCB, you'll see that each button is made of two traces, but that all of the buttons share one trace together.



This is the common (ground) trace. If we were to make a schematic, it would look kinda like this:

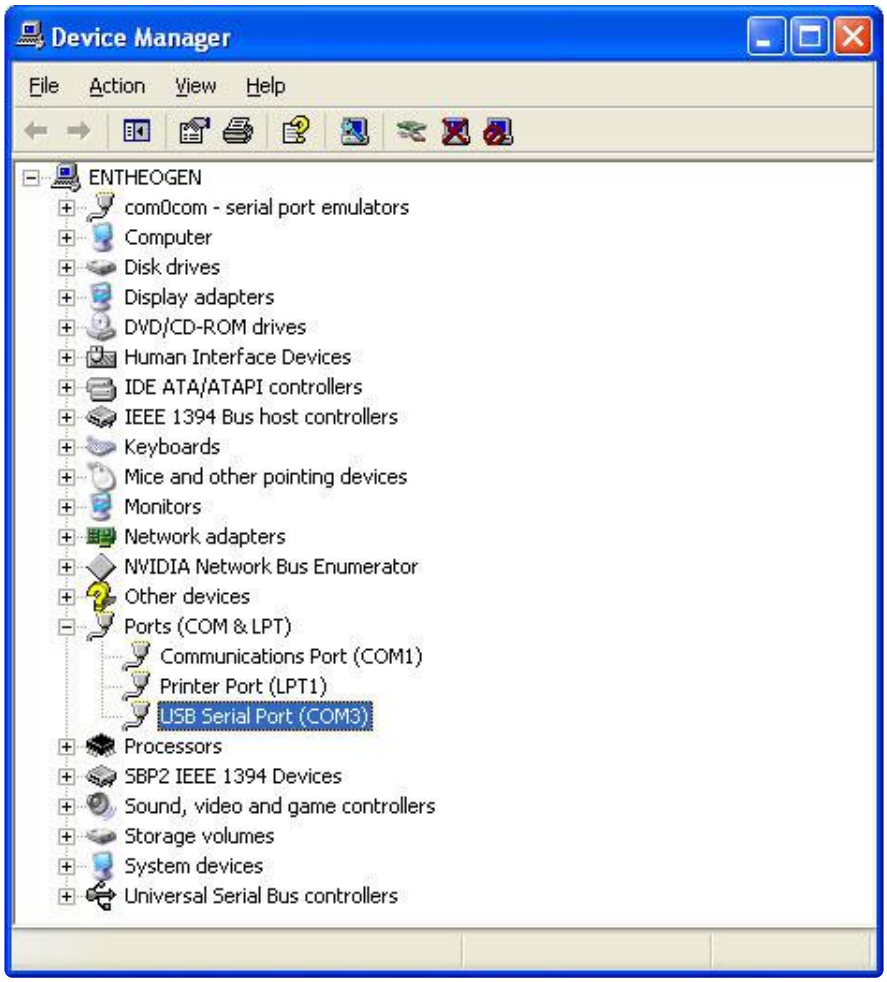


Note that this is really just a symbolic schematic, the ground wire doesn't necessarily connect on the side thats indicated, we're just showing how all the buttons have a common ground pin!

OK now this is straight forward, if you are not sure how to read buttons with a microcontroller, we have a nice tutorial you might want to check out () (in fact, we really suggest it as we'll be referring to concepts in that tutorial) Basically each button connects to an input of the microcontroller. We'll need a pull-up resistor, but luckily we can set the microcontroller's internal pullups (so we dont have to solder in 12 10K resistors!) Then the microcontroller can listen on each pin for a button press and when it is received, generate a keypress event.

Introducing the Teensy with HID

So you may be wondering "heck, I should just grab an Arduino!" But a 'proper' Arduino can't do what we want, which is to appear as a keyboard. When you plug in an Arduino into your USB port, it shows up as a Serial device, which is fantastic for debugging or for interfacing to Processing. To listen to a Serial device, you need to open up Hyperterm or Zterm or the Arduino IDE's serial monitor. However, it does not act as actual keyboard where what it outputs goes to Microsoft Word or a video game.

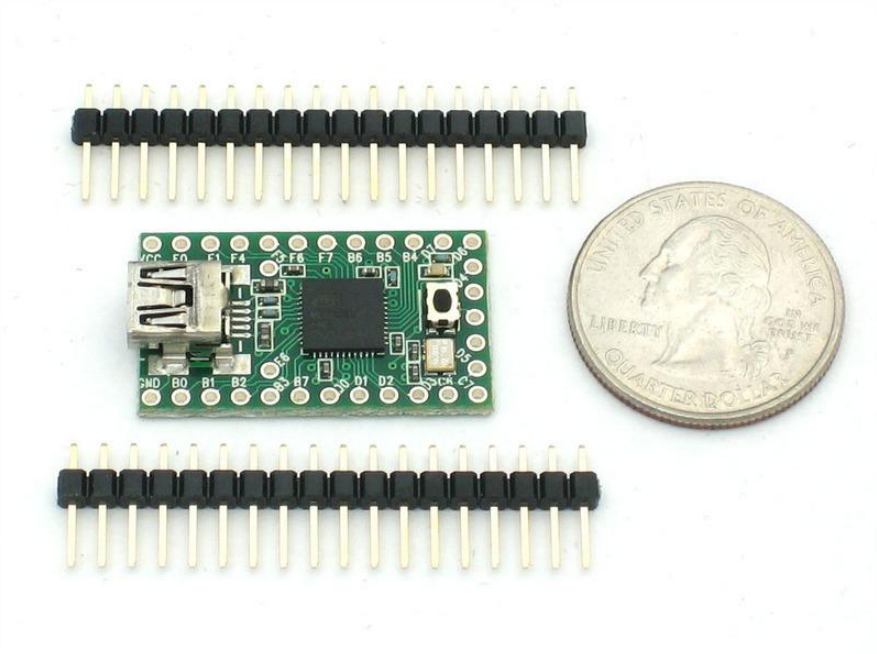


The Arduino is a USB serial port - it appears under Ports here, not under Keyboards!

For that, we need a different kind of chip, a chip that is USB native! USB native chips can act as USB serial ports, but they can also act as MIDI devices, keyboards, mice, audio devices, joysticks, etc. Nearly anything! A nice chip that does all this is the ATmega32U4 (the U is for usb!) .

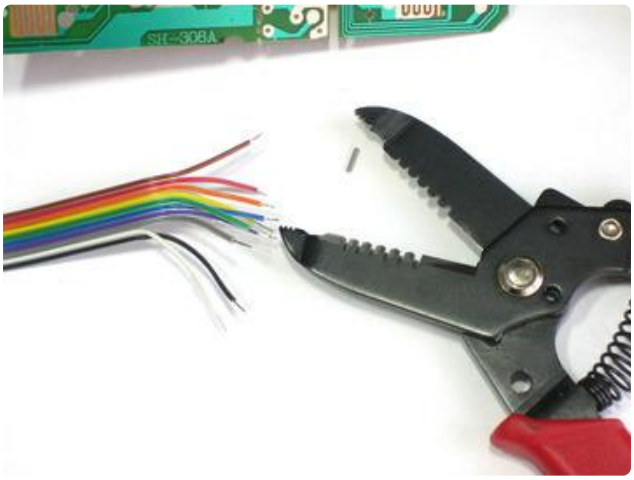
The Teensy () 2.0 is basically this chip, a USB connector, button and some other necessary things. It's very tiny (thus the name) and has a fantastic programming interface that is basically the Arduino + a helper, it runs under Mac, Linux or Windows ( ).

Since this tutorial was written, a number of other 32U4 microcontroller boards have been developed including the large Arduino Leonardo and the smaller Adafruit ItsyBitsy 32u4 - 5V 16MHz (), Adafruit ItsyBitsy 32u4 - 3V 8MHz (), and Adafruit Feather 32u4 Basic Proto (), which can do similar things.

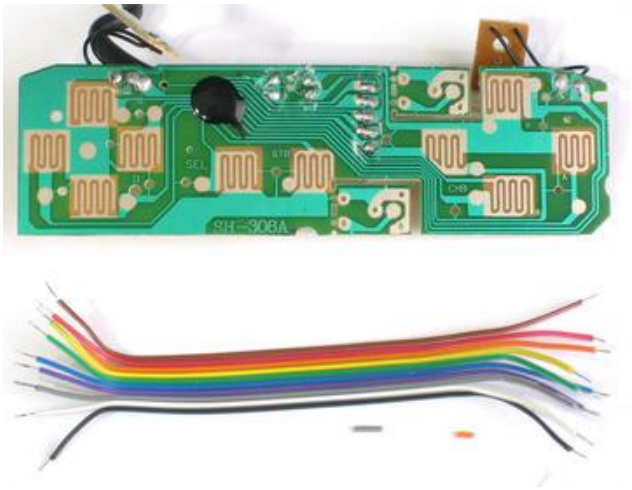


Assemble the Gamepad

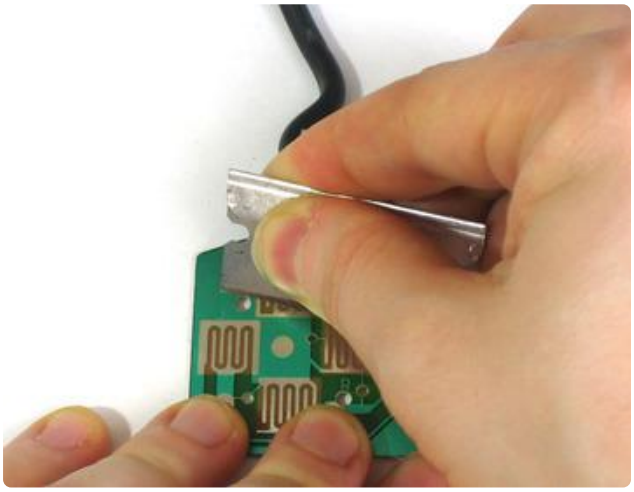
OK we're basically ready to go. The plan is to solder a single Ground wire to the common ground for all the buttons, then solder a seperate wire to each button (the not-ground side). The ground connects to the Teensy ground, the button wires connect to all the solder pads down the side. Then we'll write the code that listens to the button presses and converts them.



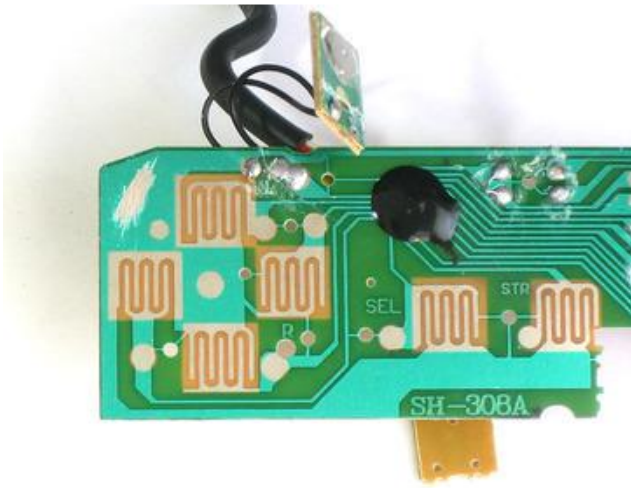
Cut off a strip of ribbon cable, about 4" long. Use diagonal cutters or fingernails to carefully nip and 'rip' the individual wires apart about 1" and then strip the ends and tin them with solder. Do this for both sides.

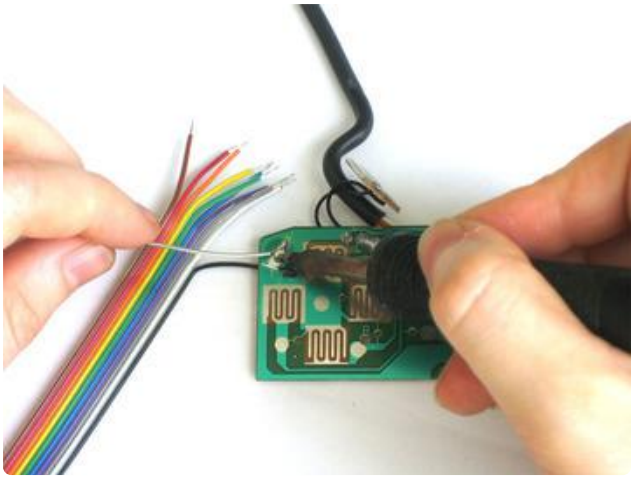


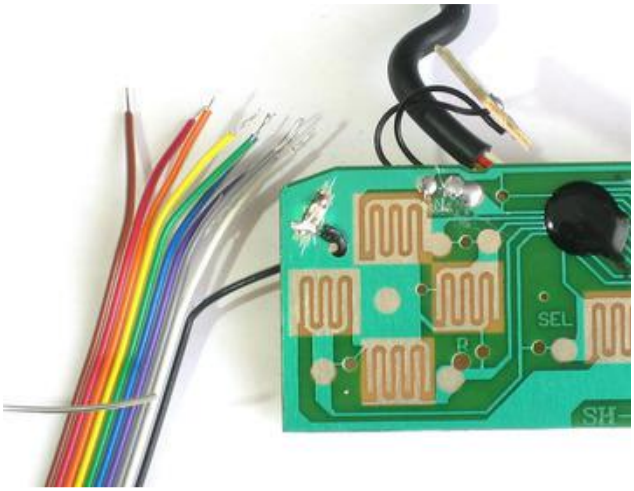
I made this cable about 1" too long initially, but its always easy to make the cable shorter!



To connect to ground, we'll expose a little copper in the top left corner, this way we don't have the wire running underneath the elastomer.





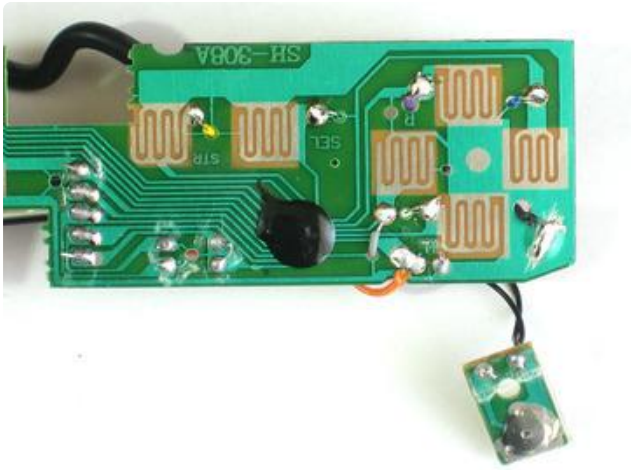


Solder the Black wire to the ground plane, we brought the wire through a hole.

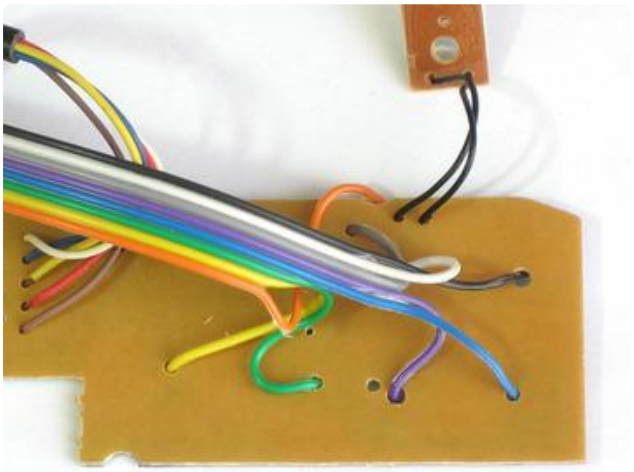
OK lets solder to the first button. The key is to remember to NOT solder to the same common pad but to the opposite pad! Solder the white wire to the 'up' button. There's almost always a hole you can feed the wire through!

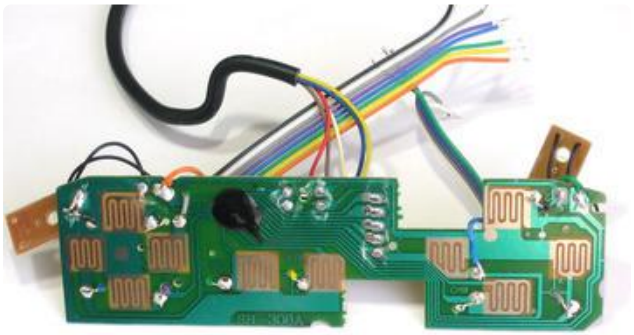
Solder the gray wire to the Right pad, the purple wire to the Down pad and the blue wire to the Left pad.

From the back.

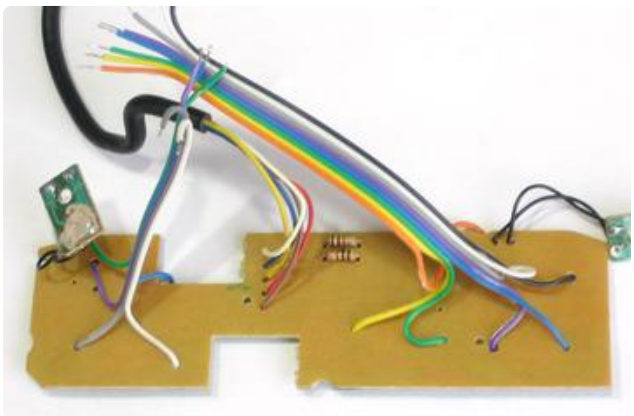


Then the orange wire goes to the L1 button, the yellow goes to Start and the green to Select.

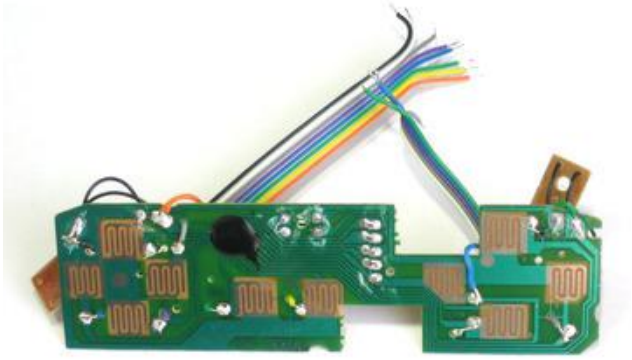




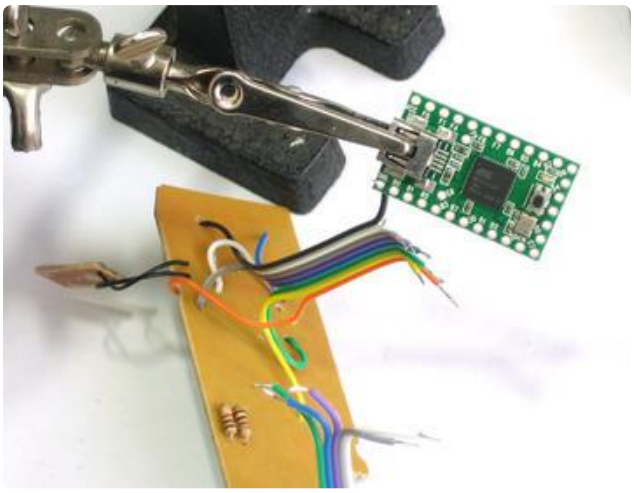
I didnt end up using the Red or Brown wires so I tore those off the ribbon. Now cut another piece the same size but with only the white, gray, purple, blue and green wires.



Connect white to B, gray to A, purple to X, blue to Y and green to R1.

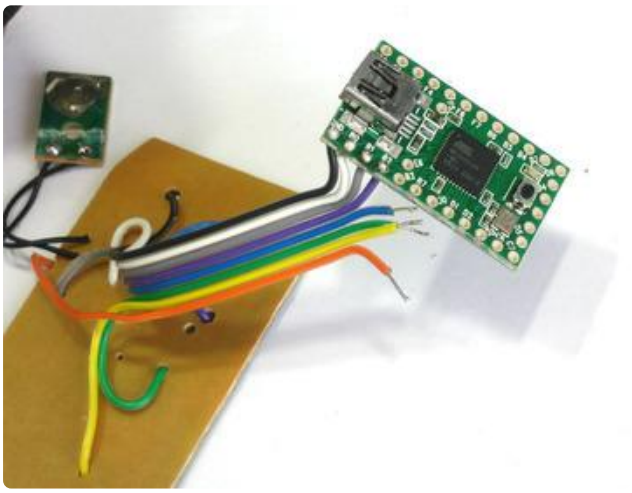


If you haven't yet, now is a good time to desolder the SNES connector cable. We wont have space for it so just pull each wire as you heat the solder joint (or just cut them short, either way).

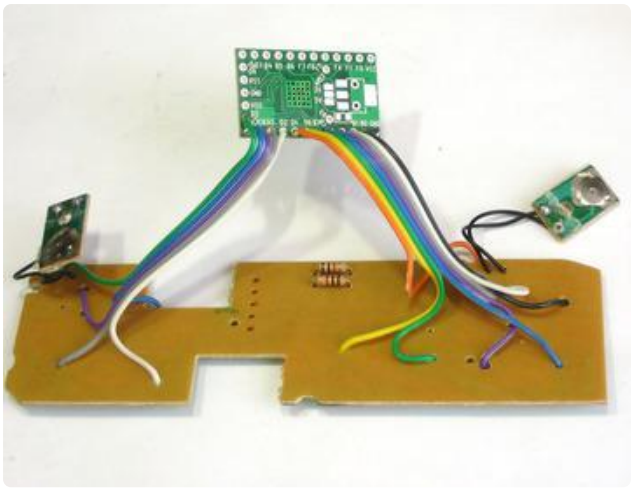


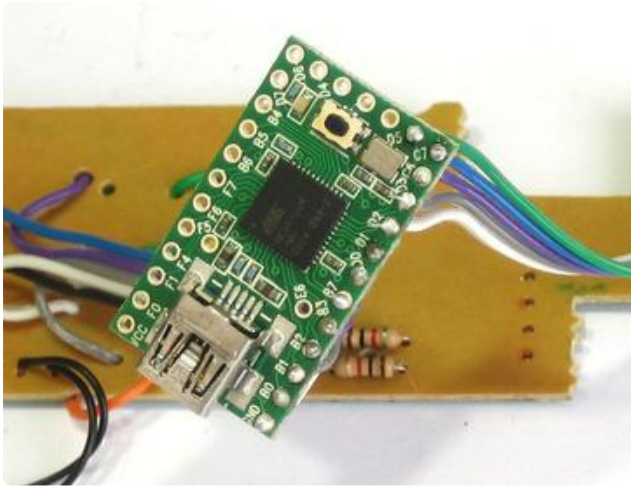
OK! Now all the buttons are wired up, its time to attach them to the Teensy. Place the Teensy in a vise or carefully use a 'third hand' to hold it (grab by the USB connector).

First, solder the black wire to the ground pin.



Next start soldering in all the ribbon cable wires, one after the other, without skipping any holes.





After the first ribbon cable, go to the second piece, starting with the white wire. The last green wire goes next to the blue one on the 'short' side.

Now we are ready to upload code to the Teensy and test out our work!

Programming the Teensy

The Teensy uses the USB connection for programing, so we don't need a seperate AVR programmer. We will use the Teensyduino IDE, which is a patch to the Arduino IDE.

If you don't have it yet, download & install the Arduino IDE software ()

Next, download the Teensyduino installer for your OS and run it, patching the Arduino IDE ()

Finally, be sure to also grab Teensyloader () which is a helper that talks to the Teensy for you.

One Button Test

We'll start with the 'one button test' sketch, which will only listen for the 'Up' D-Pad button and output the letter 'u'

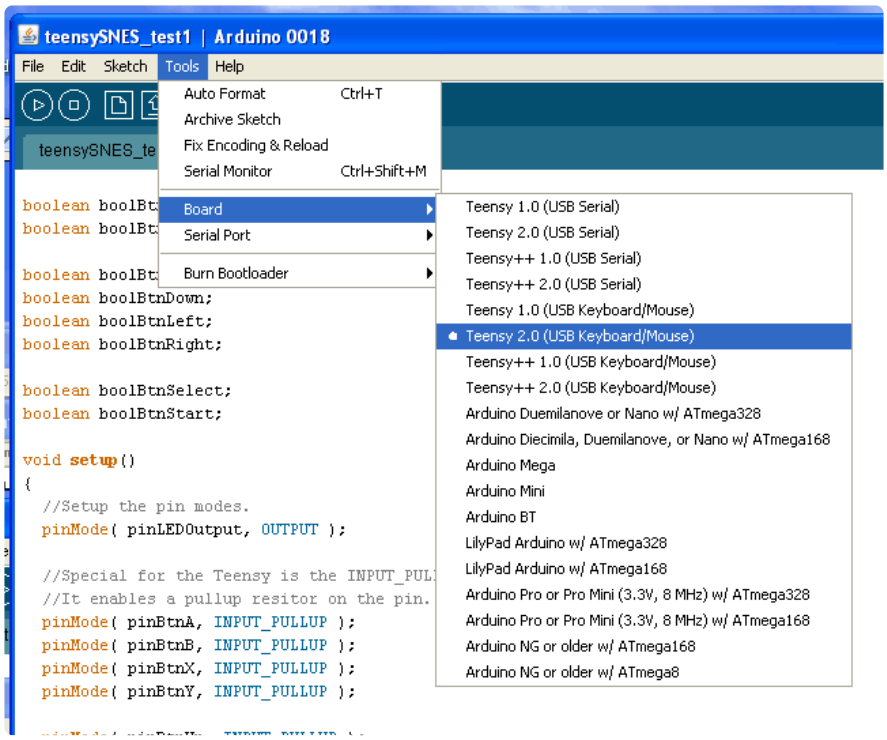
Understanding this code now will make it a lot easier to understand the later sketches that are much more complex!

You can also grab this code (which may be updated!) at GitHub ()

```
// SPDX-FileCopyrightText: 2019 Limor Fried/ladyada for Adafruit Industries
//
// SPDX-License-Identifier: MIT

const int pinBtnUp = 0;

const int pinLEDOutput = 11;

//Variables for the states of the SNES buttons
boolean boolBtnUp;


void setup()
{
  //Setup the pin modes.
  pinMode( pinLEDOutput, OUTPUT );
  //Special for the Teensy is the INPUT_PULLUP
  //It enables a pullup resitor on the pin.
  pinMode( pinBtnUp, INPUT_PULLUP );

  //Zero the SNES controller button keys:
  boolBtnUp = false;

}


void loop()
{
//  //debugging the start button...
  digitalWrite ( pinLEDOutput, digitalRead(pinBtnUp));

  //Progess the SNES controller buttons to send keystrokes.
  fcnProcessButtons();

}

//Function to process the buttons from the SNES controller
void fcnProcessButtons()
{
  //Assign temporary values for the buttons.
  //Remember, the SNES buttons are read as active LOW.
```

```
  //Capture their status here:
  boolean boolBtnUp = !digitalRead(pinBtnUp);

  if ( boolBtnUp )
  {
    //Set key1 to the U key
    Keyboard.set_key1( KEY_U );
  } else {
    Keyboard.set_key1( 0 );
  }

  //Send all of the set keys.
  Keyboard.send_now();


}
```

Now we'll upload this sketch to the Teensy. Make a new sketch and copy the code in. Select the Teensy 2.0 (USB Keyboard/Mouse) item from the Board menu.

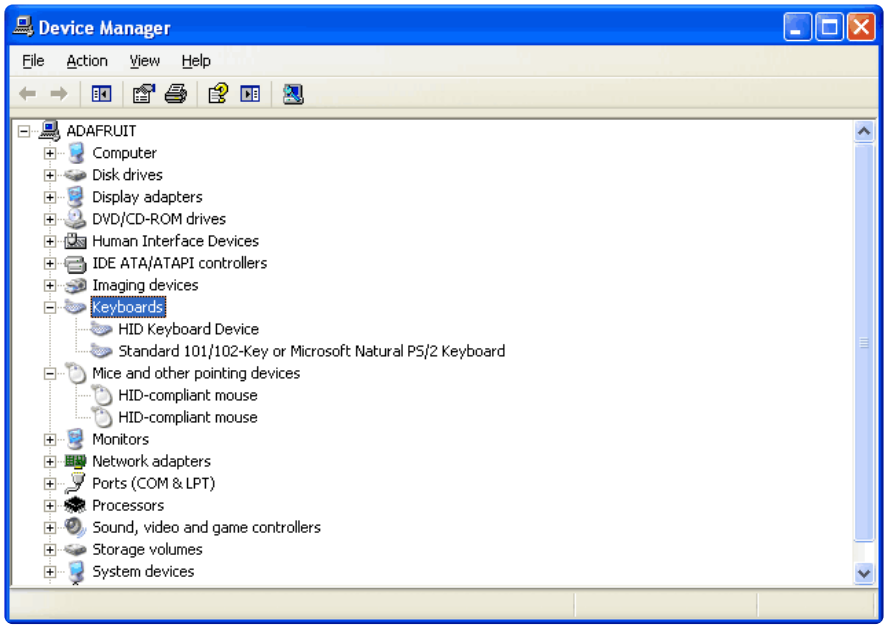


Make sure the Loader is running, if you see this:

Press the tiny button to start the bootloader, so that it looks like this:

Upload the sketch! You should see it sucessfully program the Teensy, and reboot. The OS will then alert you that it found an HID device.

And the device manager will now have an extra Keyboard and Mouse called "HID Keyboard Device" and "HID-compliant mouse"



You should now be able to open up a text editor and carefully push the 'up' D-pad to generate 'u's!

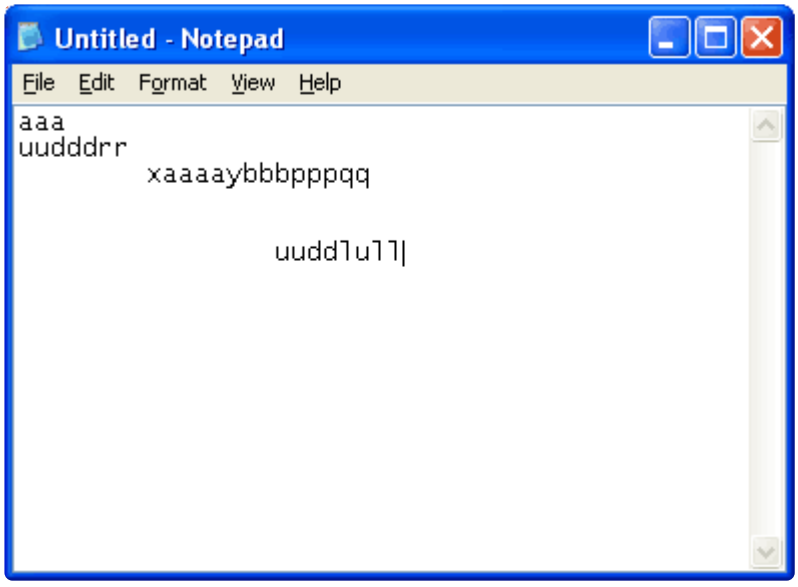
All Button Test

Next we can upload the sketch that uses all the buttons so you can test each connection. It is much longer. Download it from GitHub ().

```
// SPDX-FileCopyrightText: 2019 Limor Fried/ladyada for Adafruit Industries
//
// SPDX-License-Identifier: MIT

#define REPEATRATE 100  // milliseconds

const int pinBtnUp = 0;
const int pinBtnRight = 1;
const int pinBtnDown = 2;
const int pinBtnLeft = 3;

const int pinBtnSelect = 4;
const int pinBtnStart = 5;

const int pinBtnB = 7;
const int pinBtnA = 8;
const int pinBtnY = 10;
const int pinBtnX = 9;

const int pinBtnTrigLeft = 6;
const int pinBtnTrigRight = 23;

const int pinLEDOutput = 11;

//Variables for the states of the SNES buttons
byte buttons[] = {  pinBtnUp, pinBtnRight, pinBtnDown, pinBtnLeft, pinBtnSelect,
pinBtnStart,
                    pinBtnB, pinBtnA, pinBtnY, pinBtnX, pinBtnTrigLeft,
pinBtnTrigRight
                 };
uint16_t keys[] = {KEY_U, KEY_R, KEY_D, KEY_L, KEY_ENTER, KEY_TAB, KEY_B, KEY_A,
KEY_Y, KEY_X, KEY_P, KEY_Q};
void myset_key1(uint16_t c);
void myset_key2(uint16_t c);
void myset_key3(uint16_t c);
void myset_key4(uint16_t c);
void myset_key5(uint16_t c);
void myset_key6(uint16_t c);

#define NUMBUTTONS sizeof(buttons)

typedef void KeyFunction_t(uint16_t c);

KeyFunction_t* buttonActive[NUMBUTTONS];
KeyFunction_t* keyList[] = {myset_key6, myset_key5, myset_key4, myset_key3,
myset_key2, myset_key1};
int            keySlot = sizeof(keyList) / sizeof(KeyFunction_t*);

void setup()
{
  //Setup the pin modes.
  pinMode( pinLEDOutput, OUTPUT );

  //Special for the Teensy is the INPUT_PULLUP
  //It enables a pullup resitor on the pin.
```

```
  for (byte i=0; i< NUMBUTTONS; i++) {
    pinMode(buttons[i], INPUT_PULLUP);
  }

  //Uncomment this line to debug the acceleromter values:
//  Serial.begin();

  for (int i=0; i < NUMBUTTONS; i++) {
    buttonActive[i] = 0;
  }

}

void loop()
{
//  //debugging the start button...
  digitalWrite ( pinLEDOutput, digitalRead(pinBtnStart));

  //Progess the SNES controller buttons to send keystrokes.
  fcnProcessButtons();

}

//Function to process the buttons from the SNES controller
void fcnProcessButtons()
{
  bool keysPressed = false;
  bool keysReleased = false;

  // run through all the buttons
  for (byte i = 0; i < NUMBUTTONS; i++) {

    // are any of them pressed?
    if (! digitalRead(buttons[i]))
    {                              //this button is pressed
      keysPressed = true;
      if (!buttonActive[i])        //was it pressed before?
        activateButton(i);            //no - activate the keypress
    }
    else
    {                              //this button is not pressed
      if (buttonActive[i]) {        //was it pressed before?
        releaseButton(i);            //yes - release the keypress
        keysReleased = true;
      }
    }
  }

  if (keysPressed || keysReleased)
    Keyboard.send_now();            //update all the keypresses

}

void activateButton(byte index)
{
  if (keySlot)      //any key slots left?
  {
    keySlot--;                                  //Push the keySlot stack
    buttonActive[index] = keyList[keySlot];   //Associate the keySlot function
pointer with the button
    (*keyList[keySlot])(keys[index]);         //Call the key slot function to set
the key value
  }
}

void releaseButton(byte index)
{
  keyList[keySlot] = buttonActive[index];    //retrieve the keySlot function pointer
  buttonActive[index] = 0;                   //mark the button as no longer pressed
```

```
  (*keyList[keySlot])(0);                    //release the key slot
  keySlot++;                                 //pop the keySlot stack
}

void myset_key1(uint16_t c)
{
  Keyboard.set_key1(c);
}

void myset_key2(uint16_t c)
{
  Keyboard.set_key2(c);
}

void myset_key3(uint16_t c)
{
  Keyboard.set_key3(c);
}

void myset_key4(uint16_t c)
{
  Keyboard.set_key4(c);
}

void myset_key5(uint16_t c)
{
  Keyboard.set_key5(c);
}

void myset_key6(uint16_t c)
{
  Keyboard.set_key6(c);
}
```



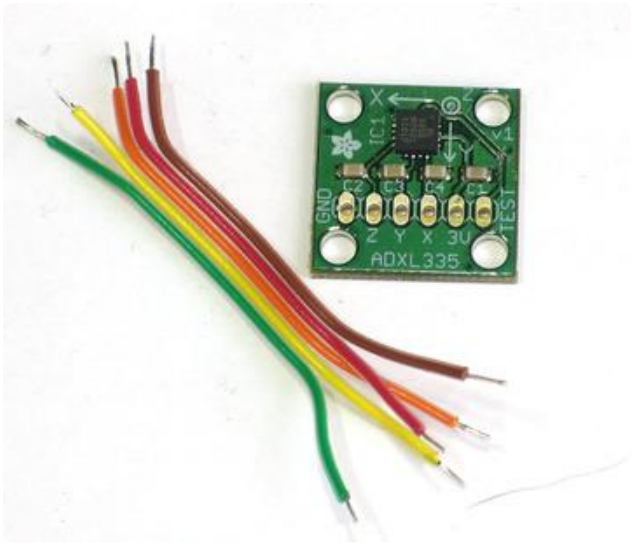
You should test all the buttons, to make sure they all output characters.

This code is more involved since it has to listen to 12 buttons. You can see at the top where we define an array of all the buttons, and then the keys that correspond to the presses. In this case, we're using a simple one-to-one correspondence for keypresses, such as Up being 'u'. To adapt this code to allow for things like "Alt-F3" would be a little more complex.

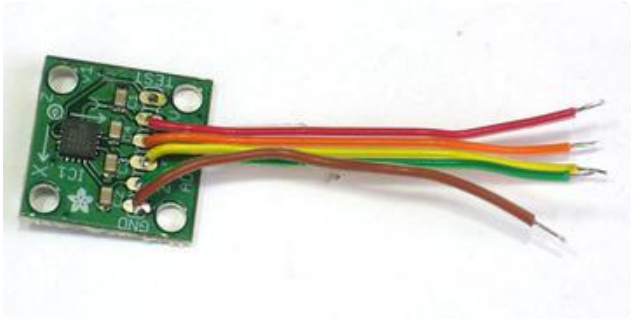
The code supports up to 6 simultaneous keypresses.

Adding the Accelerometer

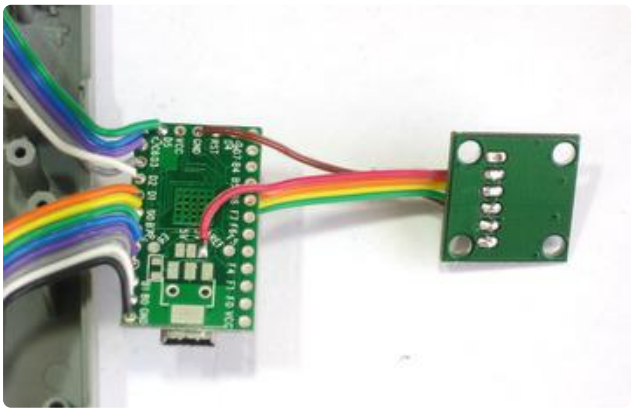
Now we will add in the accelerometer to create a tilt-activated mouse. Nearly any accelerometer will do, but the easiest to use is an analog output one. The ADXL335 will work great. First we will power the chip by providing 3.3V (not 5.0V) and ground from the Teensy, then connect the three analog outputs (X Y and Z) to three analog inputs. Finally, we will add Mouse'ing code to the sketch so that Mouse movement events are sent when the controller is tilted.



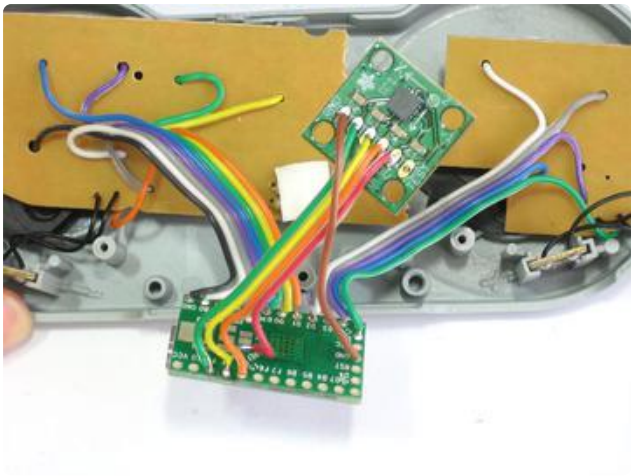
Cut a piece of ribbon cable down, we'll use Brown for Ground, Red for +3V, then Orange Yellow and Green for X Y and Z respectively.



We tore the brown wire off so that it wouldnt be twisted.



The ADXL335 requires 3V power, so don't connect it to VCC (5V) instead, we'll use the 3V that the teensy provides - it uses that voltage for the USB communication, you can't draw more than maybe 20-40mA which is plenty for this but not enough for perhaps a bunch of LEDs! Brown connects to the second GND pin.



Next connect X Y and Z to F5, F4 and F1 (don't use F0!)

You should now try out the next sketch, teensySNES_test2.ino () which will move the mouse as you tilt the controller.

```
// SPDX-FileCopyrightText: 2019 Anne Barela for Adafruit Industries
//
// SPDX-License-Identifier: MIT

const int pinAnalogXInput = 3;
const int pinAnalogYInput = 1;
const int pinAnalogZInput = 2;
const int pinAnalogDummyInput = 0;

#define KEYREPEAT 100  // milliseconds
#define KEYDELAY 200 // delay from first to second character

const int pinBtnUp = 0;
const int pinBtnRight = 1;
const int pinBtnDown = 2;
const int pinBtnLeft = 3;

const int pinBtnSelect = 4;
const int pinBtnStart = 5;

const int pinBtnB = 7;
const int pinBtnA = 8;
const int pinBtnY = 10;
const int pinBtnX = 9;

const int pinBtnTrigLeft = 6;
const int pinBtnTrigRight = 23;

const int pinLEDOutput = 11;

//Variables for the states of the SNES buttons
byte buttons[] = {  pinBtnUp, pinBtnRight, pinBtnDown, pinBtnLeft, pinBtnSelect,
pinBtnStart,
                    pinBtnB, pinBtnA, pinBtnY, pinBtnX, pinBtnTrigLeft,
pinBtnTrigRight
                 };
short keys[] = {KEY_U, KEY_R, KEY_D, KEY_L, KEY_ENTER, KEY_TAB, KEY_B, KEY_A, KEY_Y,
KEY_X, KEY_P, KEY_Q};

#define NUMBUTTONS sizeof(buttons)

typedef void KeyFunction_t(uint8_t c);

KeyFunction_t* buttonActive[NUMBUTTONS];
KeyFunction_t* keyList[] = {myset_key6, myset_key5, myset_key4, myset_key3,
```

```
myset_key2, myset_key1};
int            keySlot = sizeof(keyList) / sizeof(KeyFunction_t*);

//Change these values if accelerometer reading are different:
//How far the accerometer is tilted before
//the Teensy starts moving the mouse:
const int cintMovementThreshold = 18;

//The average zero acceleration values read
//from the accelerometer for each axis:
const int cintZeroXValue = 328;
const int cintZeroYValue = 328;
const int cintZeroZValue = 328;

//The maximum (positive) acceleration values read
//from the accelerometer for each axis:
const int cintMaxXValue = 396;
const int cintMaxYValue = 396;
const int cintMaxZValue = 396;

//The minimum (negative) acceleration values read
//from the accelerometer for each axis:
const int cintMinXValue = 256;
const int cintMinYValue = 256;
const int cintMinZValue = 256;

//The sign of the mouse movement relative to the acceleration.
//If your cursor is going in the opposite direction you think it
//should go, change the sign for the appropriate axis.
const int cintXSign = 1;
const int cintYSign = -1;
const int cintZSign = 1;

//const float cfloatMovementMultiplier = 1;

//The maximum speed in each axis (x and y)
//that the cursor should move. Set this to a higher or lower
//number if the cursor does not move fast enough or is too fast.
const int cintMaxMouseMovement = 10;

//This reduces the 'twitchiness' of the cursor by calling
//a delay function at the end of the main loop.
//There is a better way to do this without delaying the whole
//microcontroller, but that is left for another time or person.
const int cintMouseDelay = 8;


void setup()
{
    //This is not needed and set to default but can be useful if you
  //want to get the full range out of the analog channels when
  //reading from the 3.3V ADXL335.
  //If the analog reference is used, the thresholds, zeroes,
  //maxima and minima will need to be re-evaluated.
  analogReference( DEFAULT );

  //Setup the pin modes.
  pinMode( pinLEDOutput, OUTPUT );

  //Special for the Teensy is the INPUT_PULLUP
  //It enables a pullup resitor on the pin.
  for (byte i=0; i< NUMBUTTONS; i++) {
    pinMode(buttons[i], INPUT_PULLUP);
  }

  //Uncomment this line to debug the acceleromter values:
//  Serial.begin();

}
```

```
void loop()
{
//  //debugging the start button...
  digitalWrite ( pinLEDOutput, digitalRead(pinBtnStart));


  //Process the accelerometer to make the cursor move.
  //Comment this line to debug the accelerometer values:
  fcnProcessAccelerometer();

  //Progess the SNES controller buttons to send keystrokes.
  fcnProcessButtons();


  //Delay to avoid 'twitchiness' and bouncing inputs
  //due to too fast of sampling.
  //As said above, there is a better way to do this
  //than delay the whole MCU.
  delay(cintMouseDelay);
}


//Function to process the acclerometer data
//and send mouse movement information to the host computer.
void fcnProcessAccelerometer()
{
  //Initialize values for the mouse cursor movement.
  int intMouseXMovement = 0;
  int intMouseYMovement = 0;

  //Read the dummy analog channel
  //This must be done first because the X analog channel was first
  //and was unstable, it dropped or pegged periodically regardless
  //of pin or source.
  analogRead( pinAnalogDummyInput );

  //Read accelerometer readings
  int intAnalogXReading = analogRead(pinAnalogXInput);
  int intAnalogYReading = analogRead(pinAnalogYInput);
  int intAnalogZReading = analogRead(pinAnalogZInput);

  //Calculate mouse movement
  //If the analog X reading is ouside of the zero threshold...
  if( cintMovementThreshold < abs( intAnalogXReading - cintZeroXValue ) )
  {
    //...calculate X mouse movement based on how far the X acceleration is from its
zero value.
    intMouseXMovement = cintXSign * ( ( ( (float)( 2 * cintMaxMouseMovement ) / (
cintMaxXValue - cintMinXValue ) ) * ( intAnalogXReading - cintMinXValue ) ) -
cintMaxMouseMovement );
    //it could use some improvement, like making it trigonometric.
  }
  else
  {
    //Within the zero threshold, the cursor does not move in the X.
    intMouseXMovement = 0;
  }

  //If the analog Y reading is ouside of the zero threshold...
  if( cintMovementThreshold < abs( intAnalogYReading - cintZeroYValue ) )
  {
    //...calculate Y mouse movement based on how far the Y acceleration is from its
zero value.
    intMouseYMovement = cintYSign * ( ( ( (float)( 2 * cintMaxMouseMovement ) / (
cintMaxYValue - cintMinYValue ) ) * ( intAnalogYReading - cintMinYValue ) ) -
cintMaxMouseMovement );
    //it could use some improvement, like making it trigonometric.
```

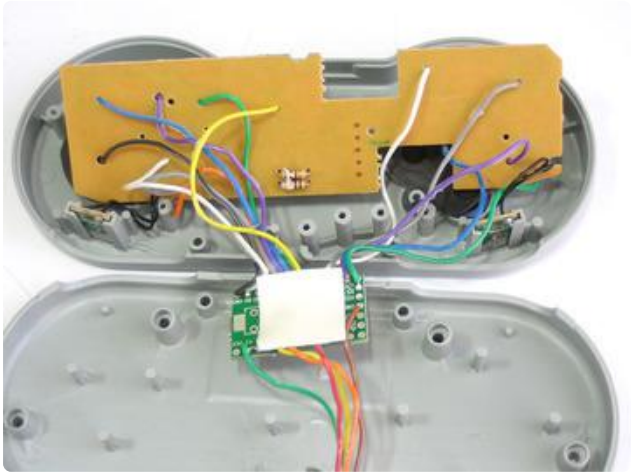
```
  }
  else
  {
    //Within the zero threshold, the cursor does not move in the Y.
    intMouseYMovement = 0;
  }

  Mouse.move(intMouseXMovement, intMouseYMovement);

}

//Function to process the buttons from the SNES controller
void fcnProcessButtons()
{
  bool keysPressed = false;
  bool keysReleased = false;

  // run through all the buttons
  for (byte i = 0; i < NUMBUTTONS; i++) {

    // are any of them pressed?
    if (! digitalRead(buttons[i]))
    {                              //this button is pressed
      keysPressed = true;
      if (!buttonActive[i])        //was it pressed before?
        activateButton(i);           //no - activate the keypress
    }
    else
    {                              //this button is not pressed
      if (buttonActive[i]) {        //was it pressed before?
        releaseButton(i);            //yes - release the keypress
        keysReleased = true;
      }
    }
  }

  if (keysPressed || keysReleased)
    Keyboard.send_now();            //update all the keypresses

}

void activateButton(byte index)
{
  if (keySlot)      //any key slots left?
  {
    keySlot--;                                  //Push the keySlot stack
    buttonActive[index] = keyList[keySlot];   //Associate the keySlot function
pointer with the button
    (*keyList[keySlot])(keys[index]);         //Call the key slot function to set
the key value
  }
}

void releaseButton(byte index)
{
  keyList[keySlot] = buttonActive[index];    //retrieve the keySlot function pointer
  buttonActive[index] = 0;                   //mark the button as no longer pressed
  (*keyList[keySlot])(0);                    //release the key slot
  keySlot++;                                 //pop the keySlot stack
}

void myset_key1(uint8_t c)
{
  Keyboard.set_key1(c);
}

void myset_key2(uint8_t c)
{
  Keyboard.set_key2(c);
```

```
}

void myset_key3(uint8_t c)
{
  Keyboard.set_key3(c);
}

void myset_key4(uint8_t c)
{
  Keyboard.set_key4(c);
}

void myset_key5(uint8_t c)
{
  Keyboard.set_key5(c);
}

void myset_key6(uint8_t c)
{
  Keyboard.set_key6(c);
}
```

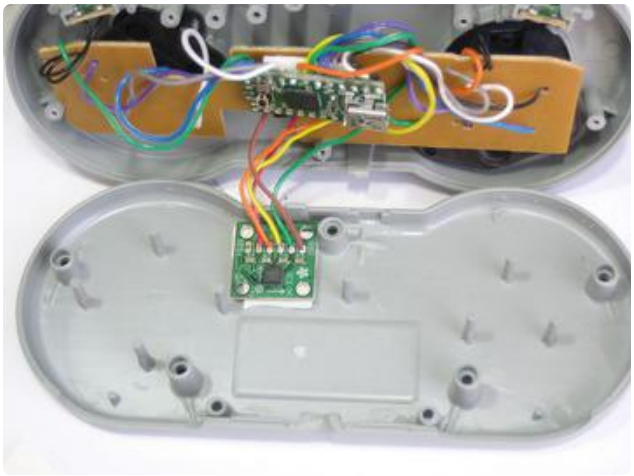
Closing it Up

Now that the mouse and keyboard are working, we can close up the game pad. This is actually the toughest part of the project, as the enclosure has plastic standoffs that are in the way.

One thing that will help is 'deribboning' the ribbon cable, so that it is easy to push around the wires.



Use sticky foam tape or hot glue to place the Teensy right at the top.



Likewise, align the acellerometer so that it is as shown (otherwise you may have some flipped axes. You should put it near the middle but we didn't see any difference being in this location.

Finally, twist the USB cable so that it goes through the strain relief posts. If this makes it really tough to close you can probably skip it and just be careful not to yank!

As you close the case, use tweezers to poke the wires around inside so that they do not interfere with the standoffs.

We wanted to make sure we could update the code without going through the disassembly process, so we drilled a hole in the back right over where the button is, then used a paper clip to push the tiny button. You can also just solder two wires to GND and RST and bring these out of the case, when shorted it will start the bootloader.