Making Video Game Controllers with Teensy

A Practical Overview

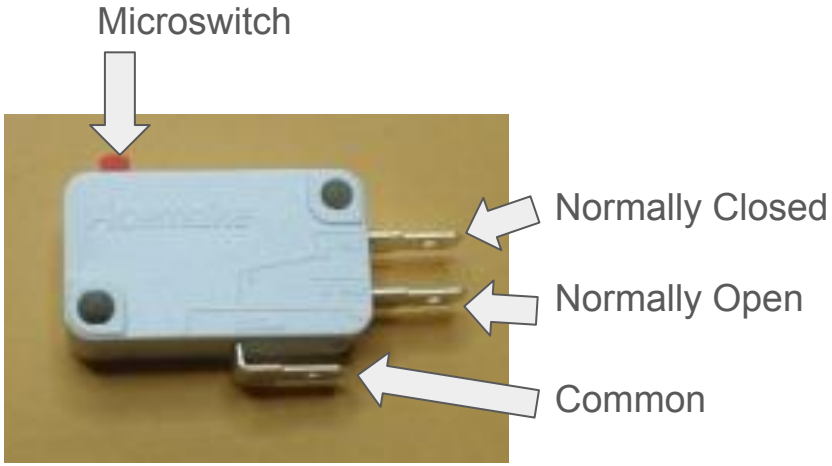
Who am I and why should you trust me about this?

- Tim Anderson (kiyoshigawa)
- Mechanical Engineering B.S. Degree from University of Utah.
- Founding / Active member of several Utah Hackerspaces.
  - HackSLC (2009-2010)
  - The Transistor (2010-2013)
  - MakeSLC (2011-2014)
  - 801 Labs (2012-Present)
- Self-taught electronics, PCB design, and programming background.
- I’ve made several working USB game controllers with the Teensy platform.

General Class Overview

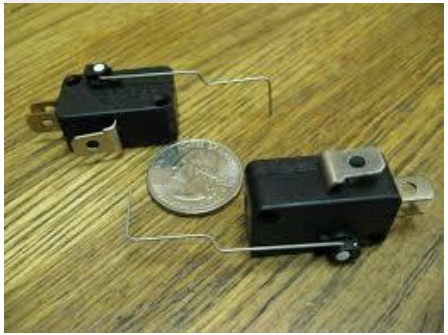
- Input Devices: Common devices used as inputs and how to connect them.
- Teensy: Why I chose this MCU, and what its libraries offer out of the box.
- General best practices for low latency responsive controller code.
- Explaining specific example code taken from my prior projects.

Input Device: Button/Switch

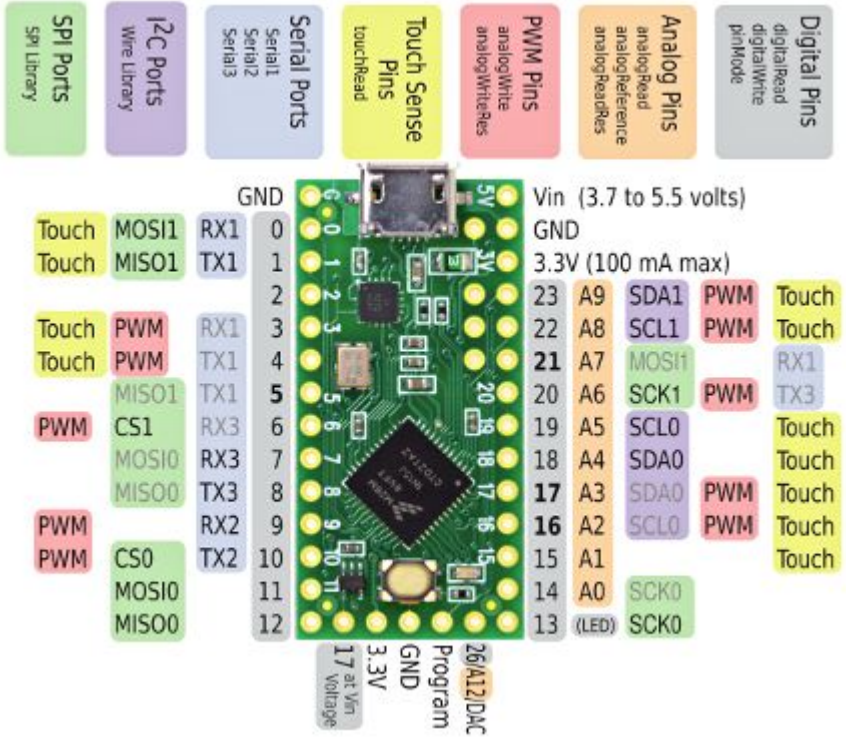


Input Device: Button/Switch

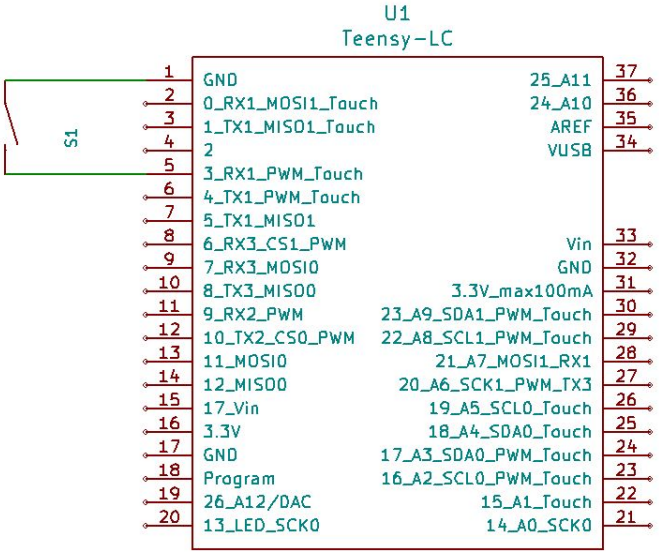
Many types of switches, same basic idea.



Input Device: Button/Switch - Connections

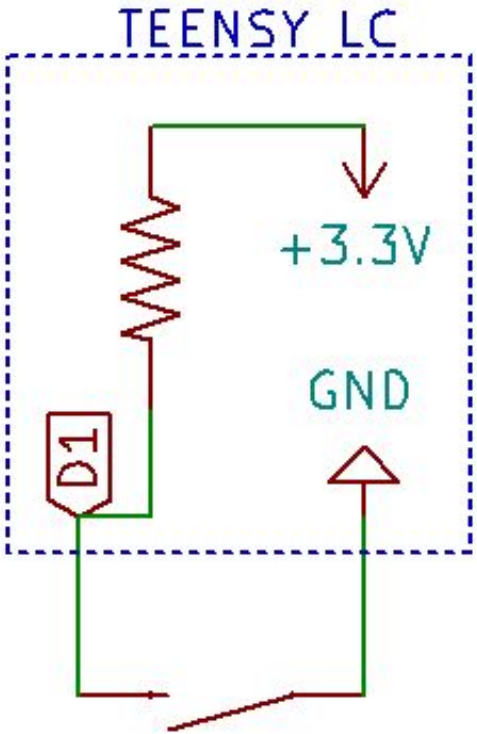


- Can be connected to any digital pin.
- Easiest type of input device to use/connect.
- Pull-up resistors are built in to the teensy.



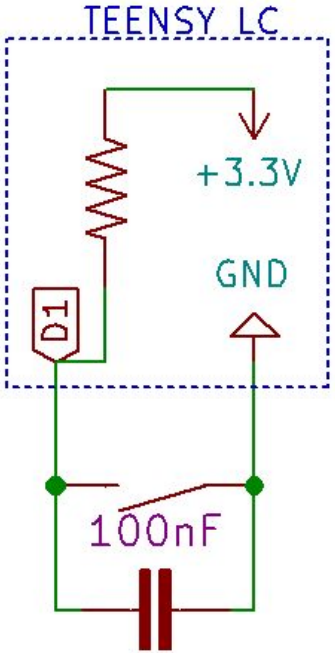
Pinout from https://www.pjrc.com/teensy/teensyLC.html

Input Device: Button/Switch - Pull-up Resistor



- Pull-up Resistors - What is it, and why do you need to know?
  - Enabled in Software on MCU
  - Causes the digital input to be biased towards high voltage when not connected.
  - When the input is connected to ground, the circuit will be complete and the input will read low.
- By using the internal pull-up resistor on the Teensy, you can avoid having to hard wire a pull-up resistor of your own.

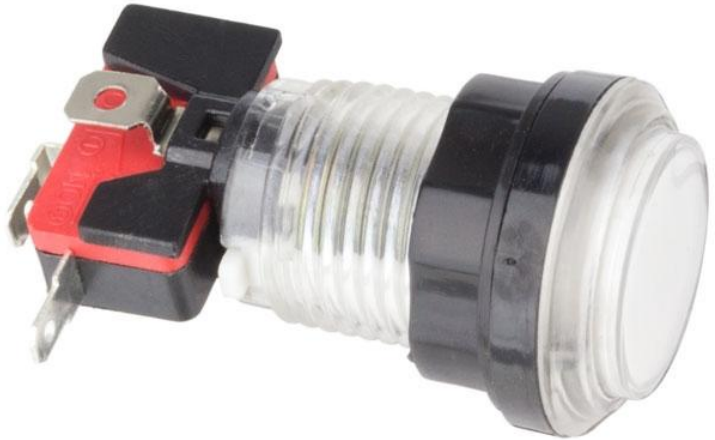
Input Device: Button/Switch - Debouncing



- What is bouncing?
  - At the moment of physical connection, a switch will rapidly ‘bounce’ between high and low state.
  - This can lead to multiple button presses registering on the MCU.
- Debouncing can be done via either hardware or software.
- Adding a capacitor between the input pin and ground is the most common, simple way to debounce via hardware.
- https://coder-tronics.com/switch-debouncing-tutorial-pt1/ This website has an awesome explaination.
- I will cover software debounce later in the presentation.

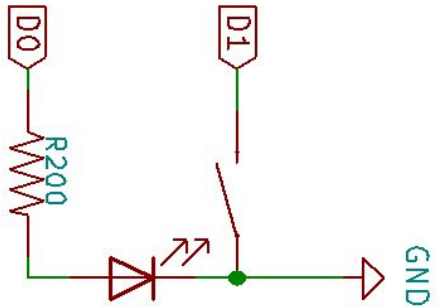
Input Device: Button/Switch with Light

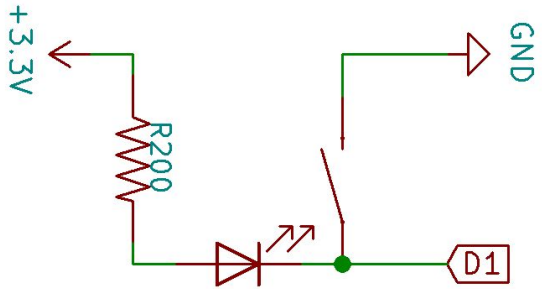
Just like a normal button, but with two extra leads for the light - typically LED these days, but they were incandescent in the past.



Input Device: Button/Switch with Light

- Can be wired to the MCU for software control - either directly to the digital pin, or through a transistor depending on current ratings.
- Can also be wired to turn on whenever the button is pressed without involving the microcontroller at all.
- Take power and current requirements into account.
  - 500mA max over USB, 100mA max through Teensy pins.



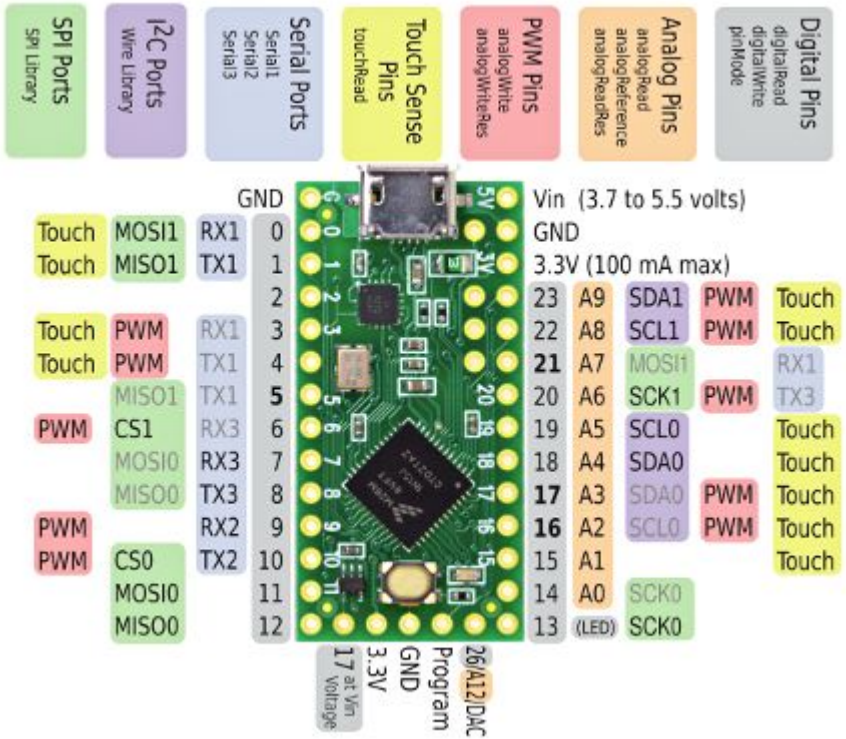


Input Device: 8-Way Joystick

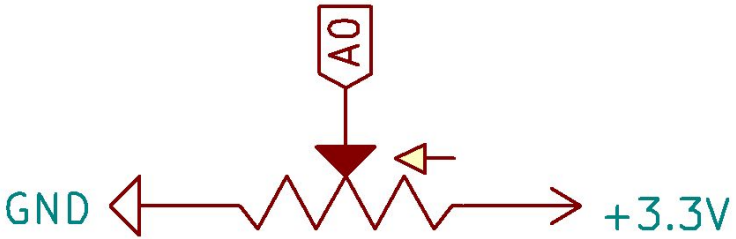
- Your standard arcade game joystick.
- Actually just 4 buttons with a stick that presses up to two at a time.
- Wired the same as a normal button, just 4 times over.

Input Device: Analog Inputs

Input Device: Analog Input



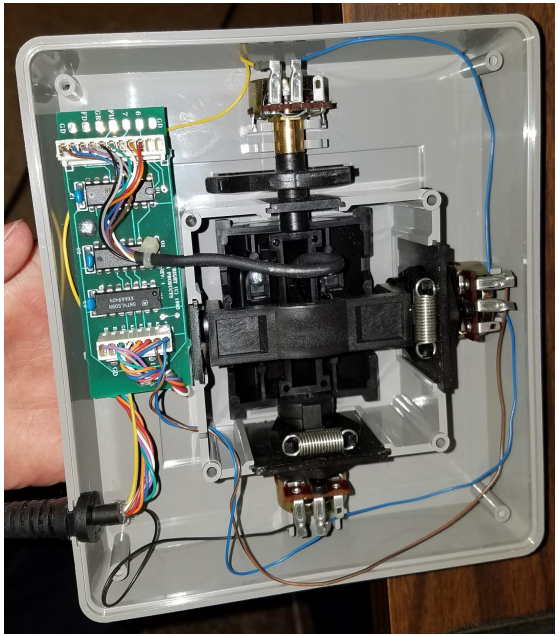
- Can be connected to any Analog Pin.
- Allows for a wide variety of input sensors.
  - Potentiometers
  - Load Cells (with supporting circuitry)
  - Microphones (with supporting circuitry)
- Need to give signals from 0V to 3.3V to be useful on a teensy - May require additional hardware to get to the right range.

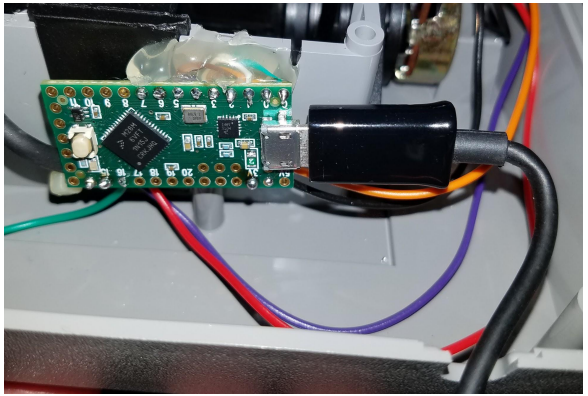


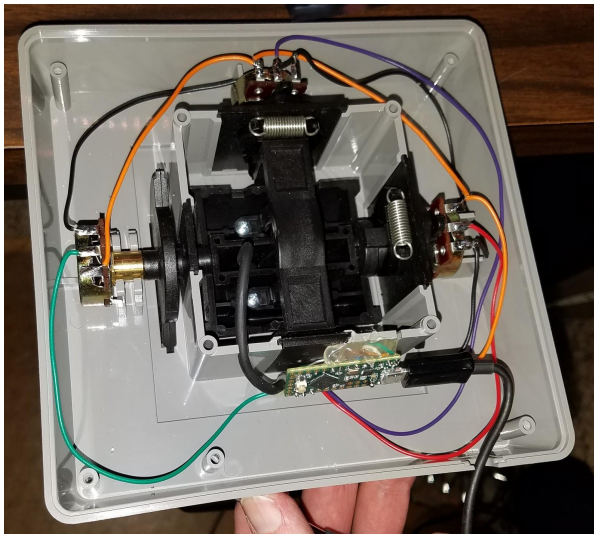
Pinout from https://www.pjrc.com/teensy/teensyLC.html

Input Devices: Analog Input - Joystick Rewiring

Pics of rewired CH Joystick to convert from Gameport to USB:

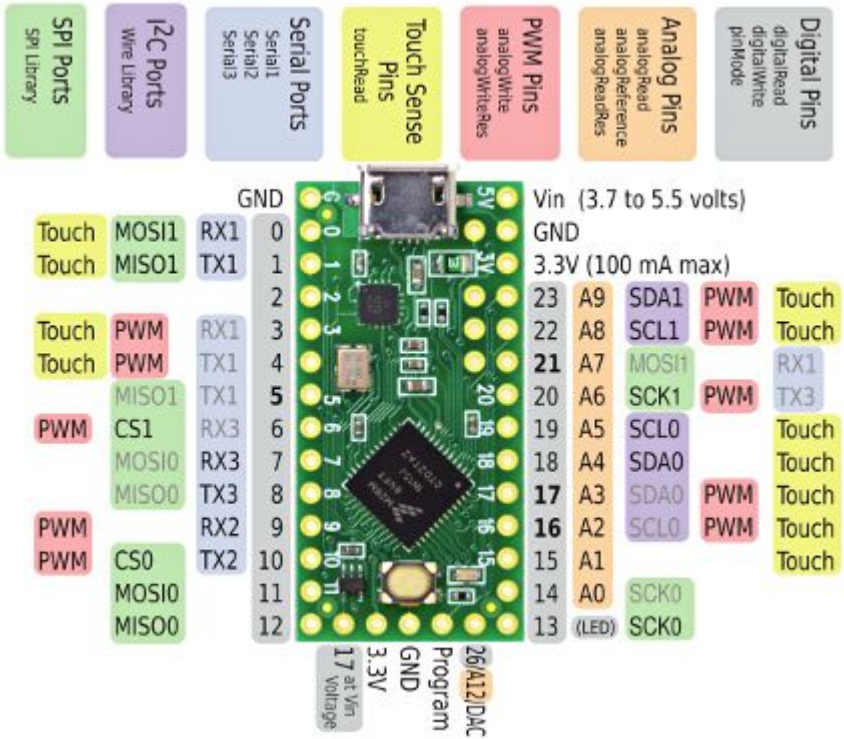






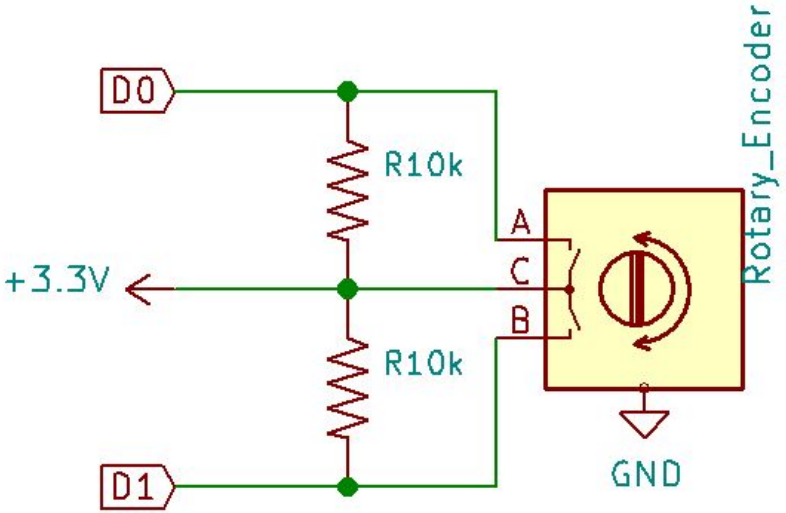
Input Device: Encoders

Input Device: Encoders



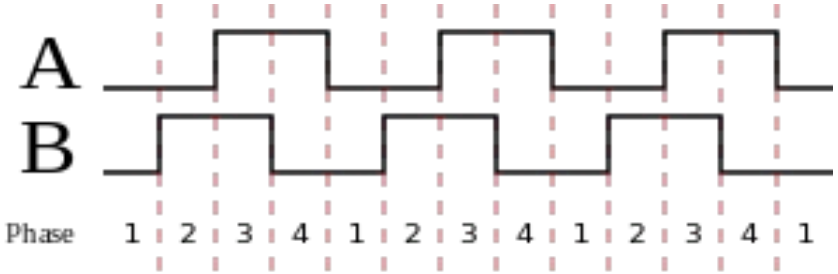
Pinout from https://www.pjrc.com/teensy/teensyLC.html

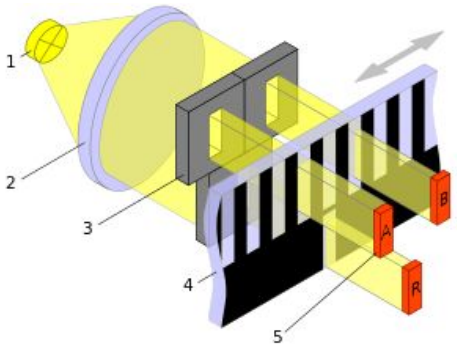
- Digital devices for measuring rotation.
- Usually needs to be powered by the board.
- All Teensy pins have a hardware interrupt.
- Teensy has a library for encoders built in.
- Different types of Encoders are available,
  - Will focus on Quadrature Encoding

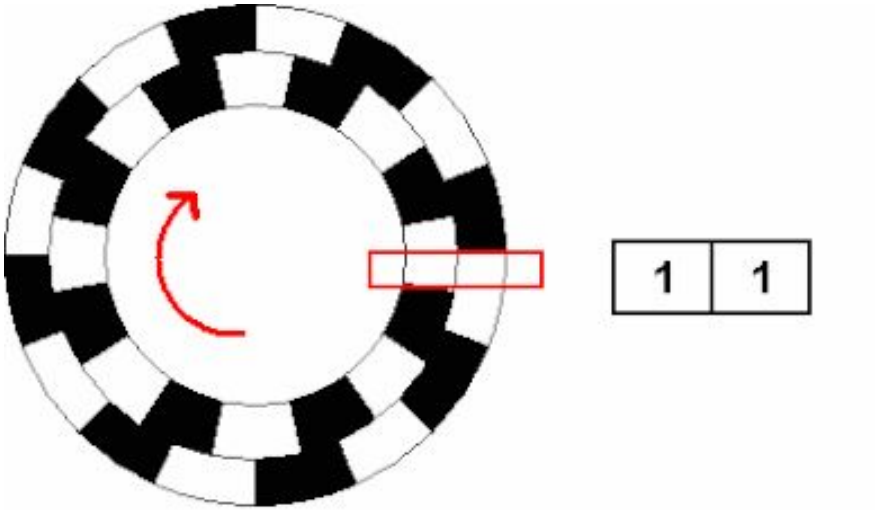


Input Device: Encoders - Quadrature Encoding

Pics stolen from wikipedia: https://en.wikipedia.org/wiki/Incremental_encoder#Quadrature_outputs

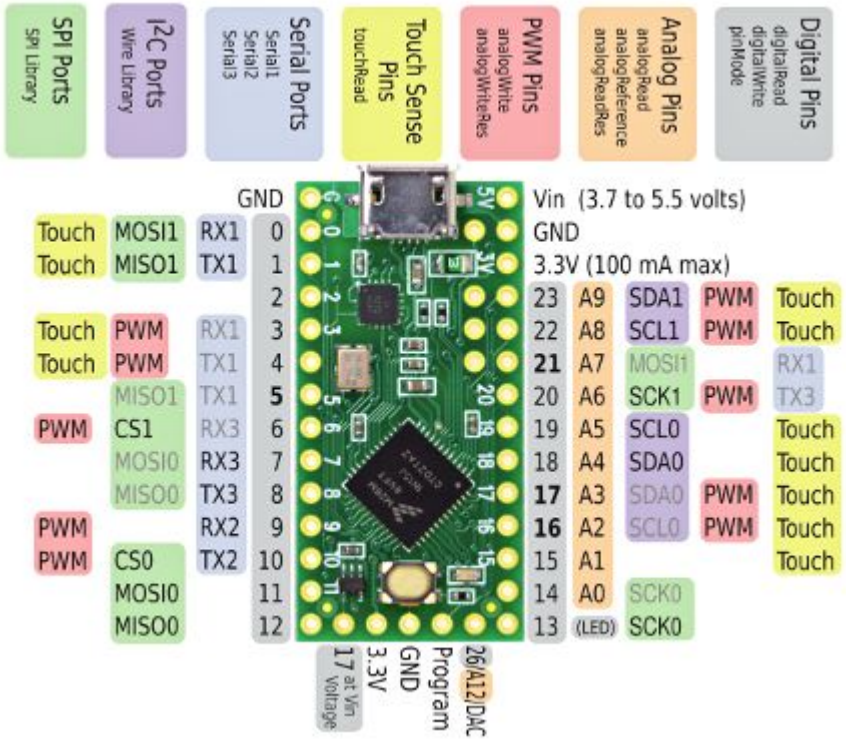




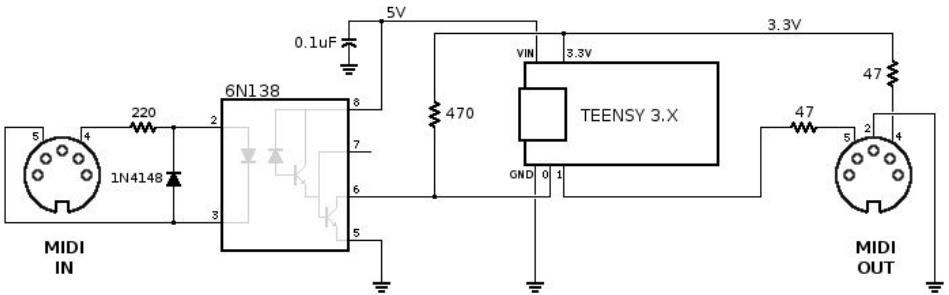


Input Device: MIDI

Input Device: MIDI

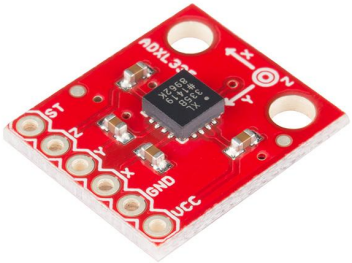


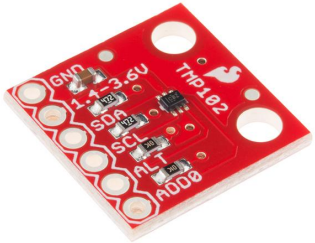
- MIDI is actually just 5V Serial at 31250 Baud.
- You WILL need to step down to 3.3V Logic for the Teensy LC - 3.x versions are often 5V tolerant - Check the site before connecting!
- A bit more complicated to connect than previous inputs.
- Can connect to any Serial Port. (LC has 3)
- Any MIDI Device can be used with the built-in libraries.



Pinout from https://www.pjrc.com/teensy/teensyLC.html

Input Device: Other





Teensy: Library Capabilities

- Teensy is an Arduino compatible microcontroller brand made by PJRC.
  - https://www.pjrc.com/teensy/
- They come in several varieties.
  - The 2.x series run on Atmel chips, and are generally slower and have less features.
  - The LC and 3.x series run on ARM Cortex-M chips and are faster and have more features.
  - I recommend 3.x for use in game controllers currently, due to price, features, and speed.
- Teensy provides many HID libraries with their software that make USB easy.
  - Keyboard
  - Joystick
  - Mouse
  - Flight Sim
  - MIDI

Teensy: Setup

Going to external sites for this part:

First install the Arduino IDE:

https://www.arduino.cc/en/Main/Software

Then install Teensyduino Software:

https://www.pjrc.com/teensy/td_download.html

https://www.pjrc.com/teensy/teensyduino.html

Teensy: Keyboard Library

Official Library Reference:

https://www.pjrc.com/teensy/td_keyboard.html

Features:

- Emulates full functionality of normal USB keyboard, Including media keys.
- Supports 6-key rollover at up to 500 keystrokes per second.
- Easy to customize code to press multiple keys, hold keys, or run macros.
- Helpful for older games, as you can map inputs to any key as needed.

Teensy: Joystick Library

Official Library Reference:

https://www.pjrc.com/teensy/td_joystick.html

Features:

- Emulates full functionality of standard HID joystick.
- Provides 6 axes, 32 buttons, and one 8-way hat switch.
- Allows precise control over when commands are sent to the computer.
- All inputs can be used simultaneously without the 6-key rollover limitation of the USB keyboard.

Teensy: Mouse Library

Official Library Reference:

https://www.pjrc.com/teensy/td_mouse.html

Features:

- Can move and click standard 3 mouse buttons - Somewhat limited.
- Supports absolute positioning if you define a specific screen size (sometimes).
- Can be useful in specific setups with a known screen size and interface elements.

Teensy: MIDI Library

Official Library Reference:

- https://www.pjrc.com/teensy/td_libs_MIDI.html

Features:

- WIth appropriate hardware, you can connect any MIDI device.
- Useful for music/rhythm game inputs, as well as games with lots of controls.
- You can also use Teensy to send out MIDI control information, such as MIDI lighting control signals. Could be used for lighting effects outside of games.
- Can also become a MIDI over USB device, but not while using a teensy as a joystick/keyboard/mouse.

Teensy: Flight Sim Library

Official Library Reference:

https://www.pjrc.com/teensy/td_flightsim.html

Features:

- Allows complicated flight sim setups to send inputs and display information from games on physical devices. - Supports X-Plane plugin for interfacing with games directly.
- Can be used with program-specific commands through the plugin, like virtual buttons.

Best Practices for Game Controller Code

- State-Based Programming - Avoid delay() functions.
  - The Arduino uses a simple programming setup, which has two required functions:
    - Setup() is a function that will run one time when the MCU is reset.
    - Loop() is a function that will run repeatedly immediately after the setup function. Upon reaching the end of the loop function, it will start over again at the top.
  - The first example everyone sees is blinking an LED.
    - The default blink code uses a delay function to pause between changing LED state.
    - This delay function uses 100% of your CPU calculations on noop commands until a certain number of miliseconds have passed.
  - There is a second, less popular, example for blinking an LED called ‘Blink Without Delay.’
    - This uses proper non-blocking state-based code to allow you to do other things while waiting.
  - Non-blocking state-based programs are essential for making sure your game controllers respond quickly and can handle many simultaneous inputs.

Basic Game Controller Structure:

- Before Setup():
  - Create global constants and variables for things like pin numbers, bounce time, state variables, etc.
  - This is where to make things like software debounce objects and set up external library objects for things like accelerometers, Hardware MIDI, I2C Wii nunchucks, etc.
- Setup():
  - Declare pin modes - set input pins as inputs and enable pull-up resistors where warranted.
  - Do any initialization of library objects as needed. It will depend on the library as to whether it needs a runtime initialization. Hardware MIDI will need to be initialized on the right hardware serial pins, for example.

Basic Game Controller Structure:

- Loop():
  - In the loop, you will need to update the state of your inputs and save this information to a state variable.
    - Some objects, like Bounce, have state variables built in, and only require calling an update function.
    - Others will require manual polling and storing that information in a state variable, like analog potentiometer inputs.
  - Once you've updated your input states, you can then act on that information to send commands via the Teensy USB libraries to your computer.
    - This can be in the form of keypresses, mouse movements, joystick updates, etc.
  - The loop function needs to use non-blocking code entirely. If any component is blocking, your controller will have responsiveness and latency issues.

Code Examples

- I will now go over several generic examples for reading input data, sending USB device commands, and then I'll show you the code I have running on the game controllers I have here today.
- You can find all the code examples from the next few slides on my github page, along with all these slides:
- https://github.com/kiyoshigawa/game_controller_code_examples

Code Examples: Button

single_button.ino

Code Examples: Many Buttons

multi_button.ino

Code Examples: Analog Input Polling

analog_input.ino

Code Examples: Quadrature Encoder Library

quadrature_encoder.ino

Code Examples: MIDI Inputs

serial_MIDI.ino

Code Examples: I2C Wii Ninchuck

nunchuck.ino

Code Examples: Actual Controller Code

- Stepmania Controller - simple controller with 12 buttons.
- Guitar Hero controller - simple controller with 10 buttons and lighting effects
- IIDX Deck - More complex controller with 18 buttons, 2 encoders, and high power lighting effects.
- CH Joystick - Revamped gameport joystick now outfitted for USB use.

Questions?

All code examples and these slides are available in a github repo I made for this class:

https://github.com/kiyoshigawa/game_controller_code_examples