Raspberry Pi Pico

Tutorials and how-to's for the Raspberry Pi Pico microcontroller (not the Raspberry Pi 2/3/4 or Zero!)

Tutorials

Use the Pico as Keyboard/Mouse input (DIY Makey Makey)

Want to have some quick button/mouse inputs to add interactivity to your setup? For prototyping this is often done by using a Makey Makey. But the board's quite expensive, and not incredibly versatile. Luckily you can build a very similar input device using a pico. After following the steps below the board will get recognized as a HID (keyboard, mouse, gamepad) by any computer and OS (Mac/Win/Linux).

This process consists of the following steps:

- Installing CircuitPython and copying the AdaFruit Human Interface Device (HID) libraries to the board.
- Changing the code to select the inputs you want to use. You have the following options:
  - Keyboard - any key on a physical keyboard, or combination of keys (ctrl-c and ctrl-v)
  - Mouse - buttons, scroll and mouse movement
  - Media buttons - play, pause, skip, sound level
  - Gamepad buttons (numbered buttons and analog joysticks)

This tutorial uses the first pages of the DIY mechanical keyboard on the Adafruit page.

Installing Circuitpython and HID libraries

First install Circuitpython to the Pico, then copy the libraries to the board. Make sure that the libraries match the version of circuitpython you are running.

Editing the code

Copy the code from this page and save it to your Pico using your favourite Python editor. Don't have one yet? The tutorial page uses Mu editor, but you can use Thonny as well.

As copied, this code maps the following keys to the following pins: --**Note that the code skips pin 15--**

Pin	Input

GP0	(KEY, (Keycode.GUI, Keycode.C)),
GP1	(KEY, (Keycode.GUI, Keycode.V)),
GP2	(KEY, [Keycode.THREE]),
GP3	(KEY, [Keycode.FOUR]),
GP4	(KEY, [Keycode.FIVE]),
GP5	(MEDIA, ConsumerControlCode.VOLUME_DECREMENT),
GP6	(MEDIA, ConsumerControlCode.VOLUME_INCREMENT),
GP7	(KEY, [Keycode.R]),
GP8	(KEY, [Keycode.G]),
GP9	(KEY, [Keycode.B]),
GP10	(KEY, [Keycode.UP_ARROW]),
GP11	(KEY, [Keycode.X]), # plus key
GP12	(KEY, [Keycode.Y]),
GP13	(KEY, [Keycode.Z]),
GP14	(KEY, [Keycode.I]),
GP15	**EMPTY - apparently a funky pin according to the tutorial**
GP16	(KEY, [Keycode.O]),
GP17	(KEY, [Keycode.LEFT_ARROW]),
GP18	(KEY, [Keycode.DOWN_ARROW]),
GP19	(KEY, [Keycode.RIGHT_ARROW]),
GP20	(KEY, [Keycode.ALT]),
GP21	(KEY, [Keycode.U]),

You can change the inputs to other KEY codes, other MEDIA codes or other MOUSE codes. Check the Adafruit HID library documentation to see which options for inputs you have. The list is quite long, so have fun! Gamepad takes a bit more effort (for using analog sticks) but the documentation gets you started quickly.

Wire up your Pico!

Connect your inputs to the relevant pins (pin number and GND). Pin pressdowns will only work when there is conductivity, so experiment around with what will connect and what doesn't. The simplest form is of course just to add a button to the end of the pin.

Removing the 'disk drive'

Once you've followed all steps above, the Pico with circuitpython will still be recognized as a disk drive besides the HID device. You can disable this in various ways if you want to. Simplest appears to be to add a boot file like here: 'create a file boot.py containing the following code:'

```
import storage

storage.disable_usb_drive()
```

And that's it. BUT: as the Pico is no longer a disk drive, it's no longer possible to edit the code! So make sure you do this last and back up your codes. If all else fails you can always nuke your pi to get it back to the factory state, see bottom of the page here.

Driving APA102 (Dotstar) LED strips with Pico

The Dotstar LEDs (APA102) are similar to the NeoPixels but with less timing requirements and other issues. You control them with two pins instead of one, and they seem to be a bit less widely used probably because they're more expensive than the Neopixel ones.

For a quick test we cobbled together a series of scripts that get some light effects out of a Dotstar strip, starting out from CircuitPython. You will need the adafruit_dotstar library that you can find here: https://circuitpython.org/libraries (make sure you download the right library version for your version of circuitpython).

More info on Dotstar strips and matrices here, by Adafruit. Page 39 and on concerns Circuitpython.

Setup

Make sure your Pico has Circuitpython installed. Once you've done that, copy adafruit_dotstar to the lib folder and this effects.py to your Pico. Connect the pins of the Dotstar strip to your Pico: power to power (preferrably external power, see also here), CI (usually yellow) to pin 18 and DI (usually green) to pin 19.

Run the effects.py and your strip should light up, cycling through a couple of lighting effects.

effects.py

effects.py has a couple of settings on the top of the program.

- num_pixels is the number of LEDs on your strip that you want lit. The program starts at the LED closest to the connection wires and then counts up.
- sleepy_time is the time each LED is lit for some of the effects. Short wait times lead to quick blinking or more fluent lighting effects.
- auto_write is set to False. When set to True, the program will wait until all leds are set before updating the entire strip at once. When set to False, each LED is updated individually. Which mode you need will depend on your use case, we set it to False to allow for faster lighting effects with longer strips. For strips up to ~10 LEDs you will hardly notice the difference.

Lines 23-115 define functions for the lighting effects, which are called in the main loop at 136.