Setting-Up Raspberry Pi for MIDI

This guide will show how to take a freshly installed Raspberry Pi (henceforth Pi) and have it operate as an OS-discoverable MIDI I/O device. It will also provide some examples of using various Python libraries to get MIDI data into and out of the programming environment.

First-time Setup

We tested all the examples in this document on a Pi Zero W using Raspian (Stretch with desktop, version March 2018). The first time, it is necessary to use a screen and keyboard to set the Pi up. Thereafter, use your method of choice to access the Pi's OS. However, do **not** use Ethernet over USB; keep the USB data port free for the other MIDI device. All steps are mandatory unless otherwise stated.

Update/Upgrade

Perform the update and upgrade as described here:

https://www.raspberrypi.org/documentation/raspbian/updating.md

Network Configuration (Optional)

If you are SSH'ing from another machine into the Pi, it is worthwhile giving the Pi a fixed IP address.

https://www.modmypi.com/blog/how-to-give-your-raspberry-pi-a-static-ip-address-update

It is also a good idea to add the network security settings to the Pi so that it will automatically connect to the network.

https://www.raspberrypi.org/documentation/configuration/wireless/wireless-cli.md

Set the Pi Up as a USB OTG Gadget

Much of this setup is from https://blog.gbaman.info/?p=699 and http://isticktoit.net/?p=1383

Open a terminal on the Pi and follow this procedure.

- Set the USB driver to dwc2
  - `echo "dtoverlay=dwc2" | sudo tee -a /boot/config.txt`
- Enable the dwc2 driver
  - `echo "dwc2" | sudo tee -a /etc/modules`
- Enable the ***libcomposite*** driver
  - `echo "libcomposite" | sudo tee -a /etc/modules`
- Enable the MIDI gadget
  - `echo "g_midi" | sudo tee -a /etc/modules`

Create the configuration script.

- Create the file
  - `sudo touch /usr/bin/midi_over_usb`
- Make it executable
  - `sudo chmod +x /usr/bin/midi_over_usb`
- Edit it with Nano
  - `sudo nano /usr/bin/midi_over_usb`

Paste the following into the file, making edits to the product and manufacturer strings as required.

```
cd /sys/kernel/config/usb_gadget/
mkdir -p midi_over_usb
cd midi_over_usb
echo 0x1d6b > idVendor # Linux Foundation
echo 0x0104 > idProduct # Multifunction Composite Gadget
echo 0x0100 > bcdDevice # v1.0.0
echo 0x0200 > bcdUSB # USB2
mkdir -p strings/0x409
echo "fedcba9876543210" > strings/0x409/serialnumber
echo "Your Name" > strings/0x409/manufacturer
echo "MIDI USB Device" > strings/0x409/product
ls /sys/class/udc > UDC
```

Exit Nano and save the file (Ctrl+X, Y, return).

Add a call to the script to rc.local, so that it executes on every startup.

```
sudo nano /etc/rc.local
```

Add the following line **before** “exit0”

```
/usr/bin/midi_over_usb
```

Exit Nano and save the file and reboot the Pi.

```
sudo reboot
```

List the available MIDI ports.

```
amidi -l
```

If the MIDI is configured correctly, the last command should output something similar to

```
Dir Device    Name
IO  hw:0,0    f_midi
```

Install Python Libraries

This section will explain how to install our preferred libraries for Python 2.x. If you are using Python 3.x, install ***mido*** and ***rtmidi*** using the ***pip3 install*** command. You can also install the packages for both Python versions. We have not experienced any conflicts.

Mido

Mido is an easy-to-use library for handling MIDI data. It relies on the ***rt-midi*** backend, the ***asound*** library, and ***Jack***. Input the following commands in sequence.

```
pip install mido
sudo apt-get install libasound2-dev
sudo apt-get install libjack-dev
pip install python-rtmidi
```

Do a quick Python command line check.

```
python
>>>import mido
>>>mido.get_output_names()
```

The output should show one 'Midi Through' port and one additional port. If this is the case, all is well.

Note: in Mido, the port name is the entire string enclosed in single quotes, but it is possible to truncate the name to the string before the colon. On this machine, the string is `'f_midi:f_midi 16:0'`.

For example, these two commands are equivalent:

```
port = mido.open_output('f_midi:f_midi 16:0')
```

```
port = mido.open_output('f_midi')
```

pigpio

We use the ***pigpio*** library to interface with the GPIO pins. We have found this library to be more stable and flexible than the standard method of interfacing with the Pi's hardware (RPi.GPIO). If you want to use another library, edit the code accordingly.

To install the ***pigpio*** library, follow the instructions here: http://abyz.me.uk/rpi/pigpio/download.html

Python Examples

The examples also use the ***numpy*** library's ***interp*** function as an easy way to map between two ranges, a la Arduino's ***map()*** function.

We used Reaper to send and receive data. The Pi is configured as a Hardware MIDI output in Reaper's preferences menu.

Control GPIO with Note Data (example_1_key_press.py)

This example shows how to:

- Listen for 3 specific note-on and note-off events using a simple condition
- Catch the exceptions that arise when non-note data is sent to the Pi (e.g. transport data from a sequencer)
- Map the note velocity to the PWM of the output pin

Import the relevant libraries, create the pi object from the pigpio library, and open the output port.

```
import mido
import pigpio
from numpy import interp

pi1 = pigpio.pi()
port = mido.open_input('f_midi') # open USB port
```

The try/catch block is to catch the errors that arise from other types of MIDI data being sent (e.g. transport controls etc.).

```
while True:
        try: #This filters out all non-note data
                for msg in port.iter_pending(): # if there is a message pending
                        if(msg.type == 'note_on'): # if it is Note On message
                                out = interp(msg.velocity, [0,127],[0,255]) #
scale velocity from 0-127 to 0-255
                                #filter the data by note number
                                if(msg.note == 53):
                                        pi1.set_PWM_dutycycle(2, out)
                                elif(msg.note == 55):
                                        pi1.set_PWM_dutycycle(3, out)
                                elif(msg.note == 57):
                                        pi1.set_PWM_dutycycle(4, out)

                        else: # if the message is not Note On (e.g. Note Off)
                                if(msg.note == 53):
                                        pi1.set_PWM_dutycycle(2, 0)
                                elif(msg.note == 55):
                                        pi1.set_PWM_dutycycle(3, 0)
                                elif(msg.note == 57):
                                        pi1.set_PWM_dutycycle(4, 0)

        except AttributeError as error:
                print("Error excepted")
                pass
```

Control GPIO with Mod and Pitch Wheels (example_2_wheels.py)

This example shows how to:

- Listen for Pitch and Mod Data and filter them by type
- Map the data to the PWM of the output pin

This example is similar to the above, with these message types:

- The Pitch wheel is type ***pitchwheel*** with a value of ***msg.pitch***
- The Mod Wheel is a Continuous Controller type ***control_change*** with a control parameter of ***msg.control = 1*** (the CC number) and a value of ***msg.value***

```
import mido
import pigpio
from numpy import interp

pi1 = pigpio.pi()
port = mido.open_input('f_midi') # open USB port

while True:
        try: #This filters out all non-note data
                for msg in port.iter_pending(): #if there is a message pending
                        if msg.type == 'pitchwheel': #of type pitchwheel
                                print("PITCH: ", msg.pitch)
                                out = interp(msg.pitch, [-8192,8192],[0,255])
                                pi1.set_PWM_dutycycle(2, out)
                                pi1.set_PWM_dutycycle(3, out)
                                pi1.set_PWM_dutycycle(4, out)
                        if msg.type == 'control_change' and msg.control == 1:
                                print("MOD: ", msg.value)
        except AttributeError as error:
                print("Error excepted")
                pass
```

Output MIDI Data from a GPIO Event

This example shows how to:

- Use an interrupt to detect a button press
- Send MIDI data from the Pi to another device

Open the output port, create two messages and setup the GPIO pin as an input. This example assumes there is a button tied to pin 21 so that that pin goes HIGH when the button is pressed

```
import mido
import pigpio

pi1 = pigpio.pi()
outport = mido.open_output('f_midi') # open USB port

onmess = mido.Message('note_on', note = 34, velocity = 127)
offmess = mido.Message('note_off', note = 34, velocity = 127)

buttPin = 21
pi1.set_mode(buttPin, pigpio.INPUT)
```

The following are the callback functions called when the button is pressed or released. The output ports ***send()*** function simply sends the message out of the port.

```
def buttonDown(gpio, level, tick):
        print("DOWN")
        outport.send(onmess)

def buttonUp(gpio, level, tick):
        print("UP")
        outport.send(offmess)
```

The callback listeners run in the background and do not need any more attention.

```
cb = pi1.callback(buttPin,pigpio.RISING_EDGE, buttonDown)
cb2 = pi1.callback(buttPin,pigpio.FALLING_EDGE, buttonUp)

#Just loop and do nothing
while True:
        pass
```

Playback a MIDI File

This example shows how to:

- Load a MIDI file in the programming environment
- Playback the file

This examples assumes you have a MIDI file called midi_file.mid in the same directory as your python script

```
import mido
from mido import MidiFile
from mido import MetaMessage

port = mido.open_output('f_midi')
mid = MidiFile('midi_file.mid')

while True:
        for msg in MidiFile('midi_file.mid').play():
                port.send(msg)
```