

Multiple UART: <https://forum.electro-smith.com/t/support-for-multiple-uarts/1409/21>

Change DAC address: https://github.com/jknipper/mcp4728_program_address

USB CDC Send/Transmit Example (Verified to work):

https://github.com/electro-smith/DaisyExamples/blob/master/seed/USB_CDC/USB_CDC.cpp

7800
72000252255255255152803696251036104252036110088691

Patch Schematics

https://github.com/electro-smith/Hardware/blob/master/reference/daisy_patch/ES_Daisy_Patch_Rev4.pdf

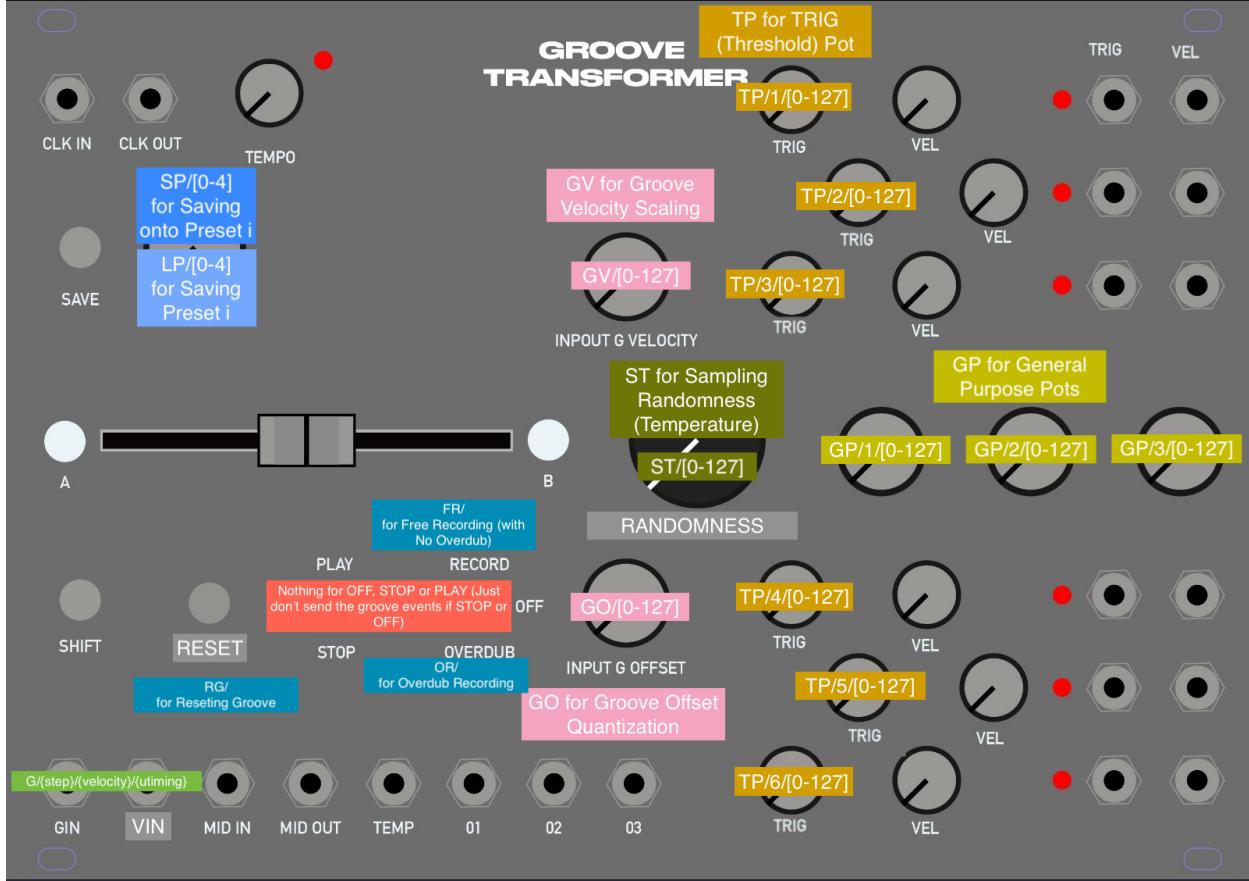
Midi In Circuit

<https://forum.electro-smith.com/t/midi-in-schamatics/971>

More examples below.

Indication	Value		
10	10 pF		
100	100 pF		
101	100 pF		
102	1000 pF	1 nF	0,001 µF
103	10 000 pF	10 nF	0,01 µF
104	100 000 pF	100 nF	0,1 µF
105	1000 000 pF	1000 nF	1 µF

DAISY<>LIBRE UART ENCODINGS

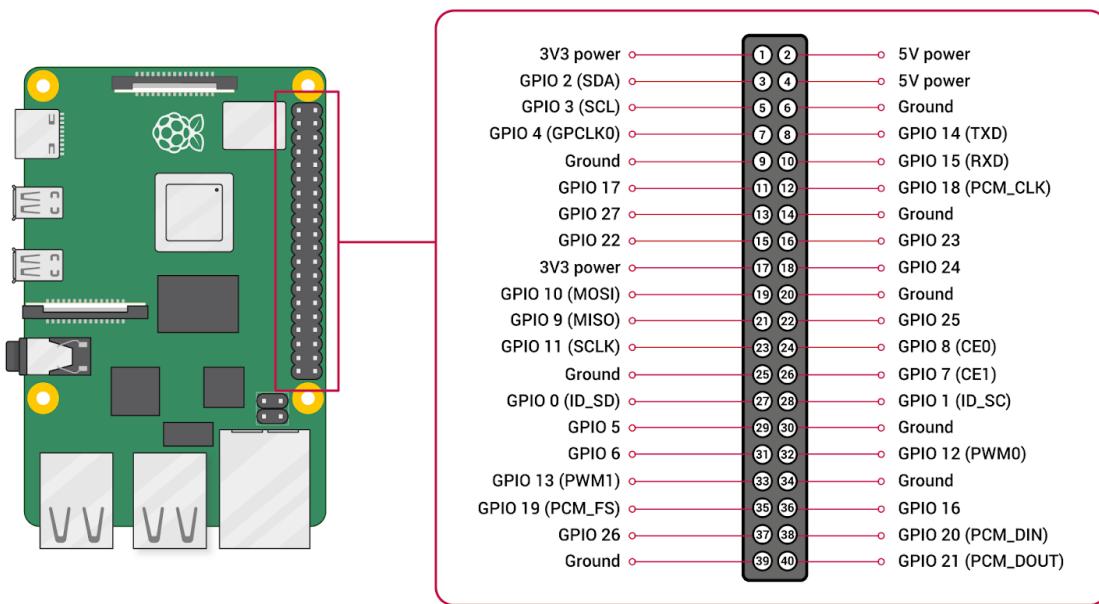


Libre Le Potato

GPIO Pins

Libre pins are almost identical to RPi (see below)

For Libre Pin Guide check out [this sheet](#)



Enabling SSH

1. sudo raspi-config

Interface Options > SSH > Enable

2. sudo if-config

get inet address → eg. 192.168.3.2

3. On pc,

ssh \$USER@192.168.3.2

SSH Via USB

Creating Image of SD Card

<https://gallaugher.com/make-a-copy-of-a-raspberry-pi-sd-card-mac/>

SERIAL Communication

```
sudo pip3 install pyserial
```

Test w/ Arduino

Setup

Tested Two Ways:

- | | |
|---------------------------------|---------------------|
| 1. Using direct usb connection— | /dev/ttyACM0 |
| 2. Using USB_TTL dongle —> | /dev/ttyUSB0 |

In either case, first should figure out the dev name. (names show up as above)

```
ls -a /dev/tty*
```

Example: Multi-threaded Python on Libre and a single module on Arduino

Py Thread 1: Asks user for an input and immediately sends to Arduino, Py Thread 2: Waits for msgs from Arduino and Prints them

Arduino: if a message is received from py, sends a confirmation message. Also, if nothing is received, reminds py that it hasn't received any messages

Pythonn	Arduino
<pre>import serial import threading ser = serial.Serial('/dev/ttyUSB0', 9600) # Change '/dev/ttyACM0' to the port that the Arduino is connected to def send_data(): while True: userInput = input("Enter a message to send to the Arduino: ") ser.write(userInput.encode()) # Send the user input to the Arduino def receive_data(): while True: arduinoData = ser.readline().decode().rstrip() # Read the incoming data from the Arduino print("Arduino sent:", arduinoData) # Print the incoming data send_thread = threading.Thread(target=send_data) receive_thread = threading.Thread(target=receive_data) send_thread.start() receive_thread.start()</pre>	<pre>unsigned long lastMsgTime; // last void setup() { Serial.begin(9600); // Set the baud rate to 9600 } void loop() { if (Serial.available() > 0) { // If data is available on the Serial port lastMsgTime = millis(); int incomingData = Serial.read(); // Read the incoming data Serial.println("received"); // Send the incoming back to Libre } if ((millis() - lastMsgTime) > 2000) { Serial.println("No Msg in two secs!"); lastMsgTime = millis(); } }</pre>

Test w/ Daisy Pod

Connection Via USB

Tested Two Ways:

3. Using direct usb connection— **/dev/ttyACM0**
4. Using USB_TTL dongle —> **/dev/ttyUSB0**
5. [Using the UART pins →](#) **/dev/ttyAML0**
[\(do this first.\)](#)

```
sudo ldtc enable uart-a
```

```
sudo ln -s /dev/ttyAML6
```

```
)
```

In either case, first should figure out the dev name. (names show up as above)

```
ls -a /dev/tty*
```


IGNORE

I2c won't be used for establishing communication between Libre and Seed as the two boards can only operate in Master mode. In an ideal case, Libre should have been the slave so that Seed pooled information every now and then at specific intervals.

Enabling I2C

(gpt - Apr 7, 2023)

To connect your Libre Le Potato to a Daisy Seed board using I2C, you'll need to follow these steps:

1. Ensure that you have the necessary software installed on your Libre Le Potato. This includes a Linux operating system and the I2C tools package.
2. Connect the two boards together using jumper wires. You'll need to connect the SDA and SCL pins on each board together, as well as connect the ground pins together.
3. On your Libre Le Potato, enable I2C by running the following command in the terminal:

```
sudo raspi-config
```

Then navigate to "Interfacing Options" > "I2C" and enable it.

4. Install the necessary Python libraries on your Libre Le Potato. You can do this by running the following commands in the terminal:

```
sudo apt-get install python3-smbus
```

5. Check if i2c is working:

```
ls /dev/i2c*
```

You should get /dev/i2c-0 and /dev/i2c-1. If only 0 shows up do the following: ([source](#))

```
ls -al /dev/i2c-*
sudo ldto enable i2c-ao
ls -al /dev/i2c-*
```

Now you should see both /dev/i2c-0 and /dev/i2c-1

6. Check if i2c is working:

```
sudo i2cdetect -y 1
```

Le Potato as Slave in I2C Communication ([Ref](#))

1. Install Dependancies

```
sudo apt install g++ pigpio
```

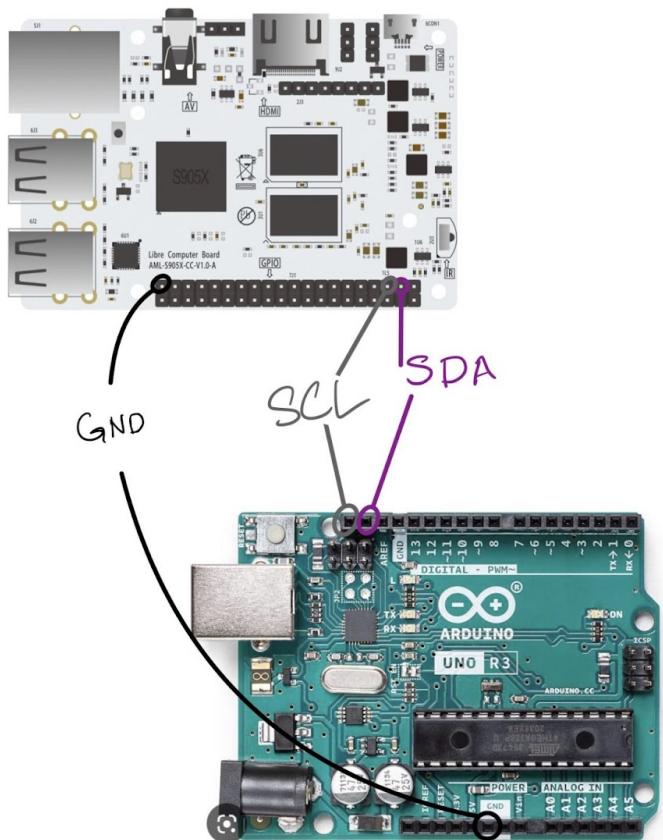
2. commented out this line in your **/boot/config.txt**:

```
dtparam=i2c_arm=on
```

- 3.

Testing I2C w/ Arduino

- . Connection



Arduino Code:

Arduino IDE under "File" -> "Examples" -> "Wire" -> "Slave Receiver".

```
void setup() {  
    Wire.begin(8); // join I2C bus with address #8  
    Wire.onReceive(receiveEvent); // register event  
    Serial.begin(9600); // start serial for output  
}
```

Le Potato

1. Check that i2c is working (i.e. 0x08 is available)

```
sudo i2cdetect -y 1
```

Output:

```
mtg@raspberrypi:~ $ sudo i2cdetect -y 1  
 0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f  
00:          --  
10:          --  
20:          --  
30:          --  
40:          --  
50:          --  
60:          --  
70:          --  
08:  ---  
mtg@raspberrypi:~ $
```

Python CODE

```
import smbus  
  
# Define I2C address of Arduino  
address = 0x04  
  
# Create I2C bus object  
bus = smbus.SMBus(1)  
  
# Write data to Arduino
```

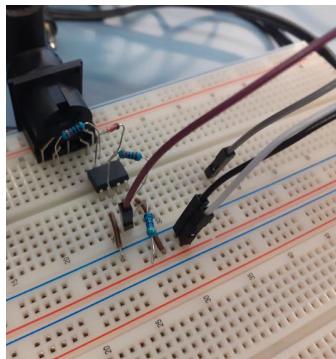
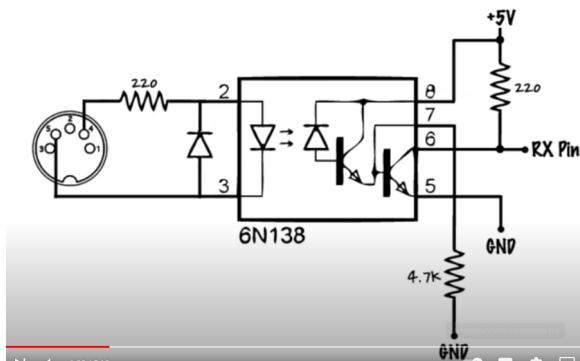
```
bus.write_byte(address, 0x01)
```

```
# Read data from Arduino  
data = bus.read_byte(address)
```

```
print("Received data:", data)
```

Midi UART

Hardware (MIDI IN)



<https://www.youtube.com/watch?v=GxfHijjn0ZM&list=LL&index=3&t=2s>
Worked on Both 3.3 and 5v Rails of Arduino

Hardware (MIDI OUT)

MIDI jack pin 5 connected to Digital pin 1 through a 220 ohm resistor

MIDI jack pin 2 connected to ground

MIDI jack pin 4 connected to +5V through a 220 ohm resistor

Software (MIDI IN, ARDUINO)

```
#include <MIDI.h> // Add Midi Library  
#define LED 13 // Arduino Board LED is on Pin 13
```

```

//Create an instance of the library with default name, serial port and settings
MIDI_CREATE_DEFAULT_INSTANCE();

void setup() {
  pinMode (LED, OUTPUT); // Set Arduino board pin 13 to output
  MIDI.begin(MIDI_CHANNEL_OMNI); // Initialize the Midi Library.
  // OMNI sets it to listen to all channels.. MIDI.begin(2) would set it
  // to respond to notes on channel 2 only.
  MIDI.setHandleNoteOn(MyHandleNoteOn); // This is important!! This command
  // tells the Midi Library which function you want to call when a NOTE ON command
  // is received. In this case it's "MyHandleNoteOn".
  MIDI.setHandleNoteOff(MyHandleNoteOff); // This command tells the Midi Library
  // to call "MyHandleNoteOff" when a NOTE OFF command is received.
  MIDI.turnThruOff();
}

void loop() { // Main loop
  MIDI.read(); // Continuously check if Midi data has been received.
}

// MyHandleNoteON is the function that will be called by the Midi Library
// when a MIDI NOTE ON message is received.
// It will be passed bytes for Channel, Pitch, and Velocity
void MyHandleNoteOn(byte channel, byte pitch, byte velocity) {
  if (pitch == 60) {
    digitalWrite(LED,HIGH); //Turn LED on
  }
}

// MyHandleNoteOFF is the function that will be called by the Midi Library
// when a MIDI NOTE OFF message is received.
// * A NOTE ON message with Velocity = 0 will be treated as a NOTE OFF message *
// It will be passed bytes for Channel, Pitch, and Velocity
void MyHandleNoteOff(byte channel, byte pitch, byte velocity) {
  if (pitch == 60) {
    digitalWrite(LED,LOW); //Turn LED on
  }
}

```

DAC MCP4728

Running Without Library

Arduino

```

#include <Wire.h> // this is the I2C library in Arduino

uint8_t DAC_1 {0x64};

#define LED 13 // arduino on board LED

bool fastWrite(
  uint16_t channel_a_value,uint16_t channel_b_value,
  uint16_t channel_c_value,uint16_t channel_d_value) {
// values must be integers between 0 and 4095
// (does not work for anything above 4095)
  uint8_t output_buffer[8];

  output_buffer[0] = channel_a_value >> 8;
  output_buffer[1] = channel_a_value & 0xFF;

  output_buffer[2] = channel_b_value >> 8;
  output_buffer[3] = channel_b_value & 0xFF;

  output_buffer[4] = channel_c_value >> 8;
  output_buffer[5] = channel_c_value & 0xFF;

```

```
output_buffer[6] = channel_d_value >> 8;
output_buffer[7] = channel_d_value & 0xFF;

for (int i=0; i<8; i++) {
    Wire.write(output_buffer[i]);
}
}

void setup() {
    Wire.begin(); // join I2C bus (address optional for master)
    pinMode (LED, OUTPUT); // Set Arduino board pin 13 to output
}

bool x = false;

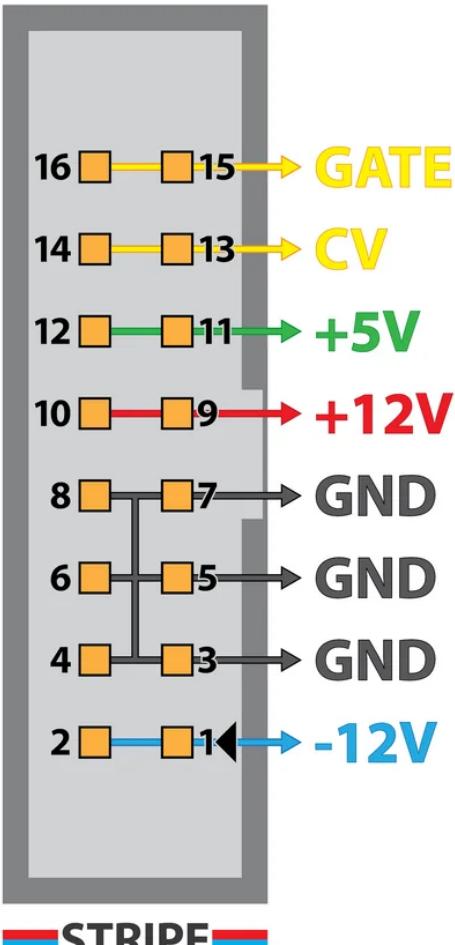
void loop() {
    Wire.beginTransmission(DAC_1); // transmit to device #8
    if (x) {
        uint16_t channel_a_value {4095};
        fastWrite(4095 , 0, 2048, 1024);
        digitalWrite(LED, HIGH);
    } else {
        fastWrite(0, 0, 0, 0);
        digitalWrite(LED, LOW);
    }

    Wire.endTransmission(); // stop transmitting
    x = !x;
    delay(1000);
}
```

Eurorack Power

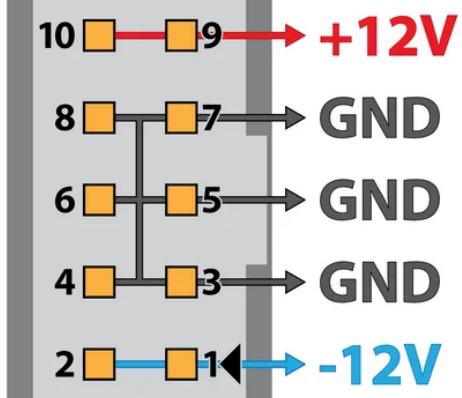
Eurorack Power Pinouts

16 PIN



STRIPE

10 PIN



STRIPE

Daisy Patch Submodule

<https://forum.electro-smith.com/t/daisy-patch-submodule-power-supply/2843>

Eurorack Dimensions

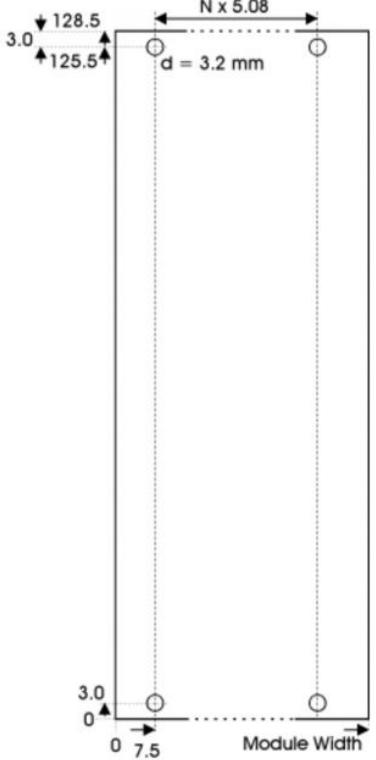
Position of the mounting holes	Module width [HP]	calculated module width [mm] (= multiples of 5.08 mm)	actual module width [mm]
	1	5.08	5.00
	1.5	7.62	7.50
	2	10.16	9.80
	4	20.32	20.00
	6	30.48	30.00
	8	40.64	40.30
	10	50.80	50.50
	12	60.96	60.60
	14	71.12	70.80
	16	81.28	80.90
	18	91.44	91.30
	20	101.60	101.30
	21	106.68	106.30
	22	111.76	111.40
	28	142.24	141.90
	42	213.36	213.00

Table 1: Position of the mounting holes and front panel width in HP and mm

DAISY WORKING EXAMPLES

ANALOG READING

```
/** Example of initializing multiple ADC inputs */
#include "daisy_seed.h"
/** This prevents us from having to type "daisy::" in front of a lot of things. */
using namespace daisy;
/** Global Hardware access */
DaisySeed hw;
int main(void)
{
    /** Initialize our hardware */
    hw.Init();
    /** Configure the ADC
     *
     * Three CV inputs (-5V to 5V -> 3V3 to 0V)
     * A0, A6, and A2
     *
     * This example was made for the Daisy Seed2 DFM, but the pins can be swapped for
     * other hardware.
     */
    AdcChannelConfig adc_cfg[3];
    adc_cfg[0].InitSingle(seed::A0);
    adc_cfg[1].InitSingle(seed::A6);
    adc_cfg[2].InitSingle(seed::A2);
    /** Initialize the ADC with our configuration */
    hw.adc.Init(adc_cfg, 3);
    /** Start the ADC conversions in the background */
    hw.adc.Start();
    /** Startup the USB Serial port */
    hw.StartLog();
    /** Infinite Loop */
    while(1)
    {
        /** Print the values via Serial every 250ms
         * Values will be 0 when inputs are 0V
         * Values will be 65536 when inputs are 3v3
         *
         * Based on the CV input circuit this means that
         * Values will be 0 when input is 5V
         * Values will be ~32768 when input is ~0V
         * Values will be 65536 when input is -5V
         */
        System::Delay(250);
        int val_0 = hw.adc.Get(0);
        // OR float val_0 = hw.adc.GetFloat(0); to get vals
        // from 0 to 1
```

```

        hw.Println("Input 1: %d", val_0);
    }
}

```

MIDI IN

```

/** Example of setting reading MIDI Input via UART
*
*
* This can be used with any 5-pin DIN or TRS connector that has been wired up
* to one of the UART Rx pins on Daisy.
* This will use D14 as the UART 1 Rx pin
*
* This example will also log incoming messages to the serial port for general MIDI
troubleshooting
*/
#include "daisy_seed.h"
/** This prevents us from having to type "daisy::" in front of a lot of things. */
using namespace daisy;
/** Fills string with string representation of MidiEvent::Type
* str needs to be at least 16 bytes long to store the data
* TODO: Move this into MIDI lib or something
*/
void GetMidiTypeAsString(MidiEvent& msg, char* str)
{
    switch(msg.type)
    {
        case NoteOff: strcpy(str, "NoteOff"); break;
        case NoteOn: strcpy(str, "NoteOn"); break;
        default: strcpy(str, "Unknown"); break;
    }
}
/** Global Hardware access */
DaisySeed      hw;
MidiUartHandler midi;
/** FIFO to hold messages as we're ready to print them */
FIFO<MidiEvent, 1024> event_log;
int counter = 0;
int main(void)
{
    /** Initialize our hardware */
    hw.Init();
    hw.StartLog();
    MidiUartHandler::Config midi_config;
    midi.Init(midi_config);
    midi.StartReceive();
    /** Infinite Loop */
    while(1)
    {
        // hw.Println("running...");
        /** Process MIDI in the background */

```

```

    midi.Listen();
    /** Loop through any MIDI Events */
    while(midi.HasEvents())
    {
        counter++;
        hw.Println("Counter: %d", counter);
        MidiEvent msg = midi.PopEvent();
        /** Handle messages as they come in
         * See DaisyExamples for some examples of this
         */
        switch(msg.type)
        {
            case NoteOn: {
                hw.Println("NOTE ON");
                // int vel = (int) msg.AsNoteOn().velocity;
                // hw.Println("Onset note vel %d", vel);
                // Do something on Note On events
                break;
            }
            default: break;
        }
        /** Regardless of message, let's add the message data to our queue to
        output */
        event_log.PushBack(msg);
    }
}

```

MIDI OUT

```

/** Example of transmitting MIDI via UART
*
*
* This can be used with any 5-pin DIN or TRS connector that has been wired up
* to one of the UART Rx pins on Daisy.
* This will use D13 as the UART 1 Tx pin
*
*
*/
#include "daisy_seed.h"
#include <array>
#include <memory>
#include <string.h>

using namespace daisy;

/** Global Hardware access */
DaisySeed      hw;
MidiUartHandler midi;
/** Fills string with string representation of MidiEvent::Type
* str needs to be at least 16 bytes long to store the data
* TODO: Move this into MIDI lib or something

```

```

*/
void MIDISendNoteOn(uint8_t channel, uint8_t note, uint8_t velocity) {
    uint8_t data[3] = { 0 };

    data[0] = (channel & 0x0F) + 0x90; // limit channel byte, add status byte
    data[1] = note & 0x7F;           // remove MSB on data
    data[2] = velocity & 0x7F;

    midi.SendMessage(data, 3);
}

void MIDISendNoteOff(uint8_t channel, uint8_t note, uint8_t velocity) {
    uint8_t data[3] = { 0 };

    data[0] = (channel & 0x0F) + 0x80; // limit channel byte, add status byte
    data[1] = note & 0x7F;           // remove MSB on data
    data[2] = velocity & 0x7F;

    midi.SendMessage(data, 3);
}

int main(void)
{
    /** Initialize our hardware */
    hw.Init();
    hw.StartLog();
    MidiUartHandler::Config midi_config;
    midi.Init(midi_config);
    midi.StartReceive();
    /** Infinite Loop */
    while(1)
    {
        System::Delay(2000);
        MIDISendNoteOn(1, 60, 60);
        System::Delay(1000);
        MIDISendNoteOff(1, 60, 60);
        hw.Println("Sending Midi..");

    }
}

```

MIDI INPUT SENT TO MIDI OUT OVER USB

```

/** Simple example of using USB MIDI
*
* When the project boots up, a 100Hz sine wave will emit from both outputs,
* and the Daisy should appear as an Audio/MIDI device on a connected host.
*

```

```

* To keep the example short, only note on messages are handled, and there
* is only a single oscillator voice that tracks the most recent note message.
*/
#include "daisy_seed.h"
#include "daisysp.h"

using namespace daisy;
using namespace daisysp;

DaisySeed      hw;
MidiUsbHandler midi;
Oscillator     osc;

void AudioCallback(AudioHandle::InputBuffer  in,
                   AudioHandle::OutputBuffer out,
                   size_t                  size)
{
    for(size_t i = 0; i < size; i++)
        out[0][i] = out[1][i] = osc.Process();
}

void MIDISendNoteOn(uint8_t channel, uint8_t note, uint8_t velocity) {
    uint8_t data[3] = { 0 };

    data[0] = (channel & 0x0F) + 0x90; // limit channel byte, add status byte
    data[1] = note & 0x7F;           // remove MSB on data
    data[2] = velocity & 0x7F;

    midi.SendMessage(data, 3);
}

void MIDISendNoteOff(uint8_t channel, uint8_t note, uint8_t velocity) {
    uint8_t data[3] = { 0 };

    data[0] = (channel & 0x0F) + 0x80; // limit channel byte, add status byte

```

```

    data[1] = note & 0x7F;           // remove MSB on data
    data[2] = velocity & 0x7F;

    midi.SendMessage(data, 3);
}

int main(void)
{
    /** Basic initialization of Daisy hardware */
    hw.Configure();
    hw.Init();

    /** Initialize USB Midi
     * by default this is set to use the built in (USB FS) peripheral.
     *
     * by setting midi_cfg.transport_config.periph =
    MidiUsbTransport::Config::EXTERNAL
     * the USB HS pins can be used (as FS) for MIDI
     */
    MidiUsbHandler::Config midi_cfg;
    midi_cfg.transport_config.periph = MidiUsbTransport::Config::INTERNAL;
    midi.Init(midi_cfg);

    /** Initialize our test tone */
    osc.Init(hw.AudioSampleRate());

    /** And start the audio callback */
    hw.StartAudio(AudioCallback);

    while(1)
    {
        /** Listen to MIDI for new changes */
        midi.Listen();

        /** When there are messages waiting in the queue... */

```

```

while(midi.HasEvents())
{
    /** Pull the oldest one from the list... */
    auto msg = midi.PopEvent();
    switch(msg.type)
    {
        case NoteOn:
        {
            /** and change the frequency of the oscillator */
            auto note_msg = msg.AsNoteOn();
            if(note_msg.velocity != 0)
                MIDISendNoteOn(10, note_msg.note, note_msg.velocity);
        }
        case NoteOff:
        {
            auto note_msg = msg.AsNoteOff();
            MIDISendNoteOff(10, note_msg.note, note_msg.velocity);
        }
        break;
        // Since we only care about note-on messages in this example
        // we'll ignore all other message types
    default: break;
    }
}
}
}
}

```

MIDI CC MESSAGES

All CC messages sent on channel 4

A Button: 4

B Button: 5

Crossfader Position: 6

Uncertainty: 7

General Purpose 1: 8

General Purpose 2: 9

General Purpose 3: 10

Groove Velocity: 11

Groove Offset: 12

Voice Velocity Scale 1: 14
Voice Velocity Scale 2: 15
Voice Velocity Scale 3: 16
Voice Velocity Scale 4: 17
Voice Velocity Scale 5: 18
Voice Velocity Scale 6: 19
Voice Density Scale 1: 20
Voice Density Scale 2: 21
Voice Density Scale 3: 22
Voice Density Scale 4: 23
Voice Density Scale 5: 24
Voice Density Scale 6: 25
Save 1: 26
Save 2: 27
Save 3: 28
Save 4: 29
Save 5: 30
Save 6: 31
Load 1: 32
Load 2: 33
Load 3: 34
Load 4: 35
Load 5: 36
Load 6: 37

DAC - FINDS ADDRESSES

```
#include "daisy_seed.h"
#include <array>
#include <memory>
#include <string.h>

using namespace daisy;
static DaisySeed hw;
unsigned char DAC_1 {0x64};
unsigned char DAC_2 {0x60};
I2CHandle::Config _i2c_config;
I2CHandle _i2c;
void Findi2c();
```

```

void Findi2c() {
    hw.Println("Scanning...");
    int nDevices = 0;
    for(unsigned char address = 1; address < 127; address++) {
        uint8_t testData = 0;
        I2CHandle::Result i2cResult = _i2c.TransmitBlocking(address, &testData, 1,
500);

        if(i2cResult == I2CHandle::Result::OK) {
            int prAddress = (address < 16) ? 0 : address;
            hw.Println("I2C device found at address %x !", prAddress);
            nDevices++;
            DAC_1 = address;
        }
    }
    if(nDevices == 0) {
        hw.Println("No I2C devices found");
    }
    else {
        hw.Println("done");
    }
}

int main(void)
{
    hw.Configure();
    hw.Init();
    hw.StartLog(true);
    System::Delay(1000);

    _i2c_config.periph = I2CHandle::Config::Peripheral::I2C_1;
    _i2c_config.speed = I2CHandle::Config::Speed::I2C_400KHZ;
    _i2c_config.mode = I2CHandle::Config::Mode::I2C_MASTER;
    _i2c_config.pin_config.scl = {DSY_GPIOB, 8};
    _i2c_config.pin_config.sda = {DSY_GPIOB, 9};
    // initialise the peripheral
    _i2c.Init(_i2c_config);

    while(1) {
        System::Delay(5000);
        hw.Println("Running..");
        Findi2c();
    }
}

```

DAC - WORKING EXAMPLE USING DMATRANSMIT

```
#include "daisy_seed.h"
#include <array>
#include <memory>
#include <string.h>

using namespace daisy;
unsigned char DAC_1 {0x64};

int main(void)
{
    static DaisySeed seed;
    seed.Configure();
    seed.Init();
    seed.StartLog(true);
    System::Delay(1000);

    I2CHandle::Config _i2c_config;
    _i2c_config.periph = I2CHandle::Config::Peripheral::I2C_1;
    _i2c_config.speed  = I2CHandle::Config::Speed::I2C_400KHZ;
    _i2c_config.mode   = I2CHandle::Config::Mode::I2C_MASTER;
    _i2c_config.pin_config.scl = {DSY_GPIOB, 8};
    _i2c_config.pin_config.sda = {DSY_GPIOB, 9};

    // initialise the peripheral
    I2CHandle _i2c;
    _i2c.Init(_i2c_config);

    while(1) {
        System::Delay(1000);
        seed.Println("Running..");

        // uint8_t output_buffer[8];
#define BUFF_SIZE 8
        static uint8_t DMA_BUFFER_MEM_SECTION output_buffer[BUFF_SIZE];
        uint16_t channel_a_value {0};
        uint16_t channel_b_value {1000};
        uint16_t channel_c_value {2000};
        uint16_t channel_d_value {4095};

        output_buffer[0] = static_cast<uint8_t>(channel_a_value >> 8);
        output_buffer[1] = static_cast<uint8_t>(channel_a_value & 0xFF);
        output_buffer[2] = static_cast<uint8_t>(channel_b_value >> 8);
        output_buffer[3] = static_cast<uint8_t>(channel_b_value & 0xFF);
        output_buffer[4] = static_cast<uint8_t>(channel_c_value >> 8);
        output_buffer[5] = static_cast<uint8_t>(channel_c_value & 0xFF);
        output_buffer[6] = static_cast<uint8_t>(channel_d_value >> 8);
        output_buffer[7] = static_cast<uint8_t>(channel_d_value & 0xFF);
        seed.Println("after filling buffer");
    }
}
```

```

        for (int i=0; i<8; i++) {
            seed.Println("Inside loop %d", i);
            I2CHandle::Result i2cResult= _i2c.TransmitDma(0x64, &output_buffer[0], 8,
NULL, NULL);
            if(i2cResult == I2CHandle::Result::OK) {
                seed.Println("OK TRANSMISSION");
            }
        }
    }
}

```

DAC - WORKING EXAMPLE USING DMATRANSMIT WITH 2 DACS OF DIFFERENT ADDRESSES

```

#include "daisy_seed.h"
#include <array>
#include <memory>
#include <string.h>

using namespace daisy;
unsigned char DAC_1 {0x64};
// uint8_t output_buffer[8];
#define BUFF_SIZE 8
static uint8_t DMA_BUFFER_MEM_SECTION output_buffer[BUFF_SIZE];

int main(void)
{
    static DaisySeed seed;
    seed.Configure();
    seed.Init();
    seed.StartLog(true);
    System::Delay(1000);

    I2CHandle::Config _i2c_config;
    _i2c_config.periph = I2CHandle::Config::Peripheral::I2C_1;
    _i2c_config.speed = I2CHandle::Config::Speed::I2C_400KHZ;
    _i2c_config.mode = I2CHandle::Config::Mode::I2C_MASTER;
    _i2c_config.pin_config.scl = {DSY_GPIOB, 8};
    _i2c_config.pin_config.sda = {DSY_GPIOB, 9};

    // initialise the peripheral
    I2CHandle _i2c;
    _i2c.Init(_i2c_config);
}

```

```

while(1) {
    seed.Println("Running..");

    System::Delay(1000);

    uint16_t channel_a_value {4000};
    uint16_t channel_b_value {2000};
    uint16_t channel_c_value {3000};
    uint16_t channel_d_value {4095};

    output_buffer[0] = static_cast<uint8_t>(channel_a_value >> 8);
    output_buffer[1] = static_cast<uint8_t>(channel_a_value & 0xFF);
    output_buffer[2] = static_cast<uint8_t>(channel_b_value >> 8);
    output_buffer[3] = static_cast<uint8_t>(channel_b_value & 0xFF);
    output_buffer[4] = static_cast<uint8_t>(channel_c_value >> 8);
    output_buffer[5] = static_cast<uint8_t>(channel_c_value & 0xFF);
    output_buffer[6] = static_cast<uint8_t>(channel_d_value >> 8);
    output_buffer[7] = static_cast<uint8_t>(channel_d_value & 0xFF);

    I2CHandle::Result i2cResult= _i2c.TransmitDma(0x64, &output_buffer[0], 8,
NULL, NULL);
    if(i2cResult == I2CHandle::Result::OK) {
        seed.Println("OK TRANSMISSION 1");
    }
    I2CHandle::Result i2cResult_2= _i2c.TransmitDma(0x60, &output_buffer[0], 8,
NULL, NULL);
    if(i2cResult_2 == I2CHandle::Result::OK) {
        seed.Println("OK TRANSMISSION 2");
    }

    System::Delay(1000);

    uint16_t channel_a_value_2 {0};
    uint16_t channel_b_value_2 {0};
    uint16_t channel_c_value_2 {0};
    uint16_t channel_d_value_2 {0};

    output_buffer[0] = static_cast<uint8_t>(channel_a_value_2 >> 8);
    output_buffer[1] = static_cast<uint8_t>(channel_a_value_2 & 0xFF);
    output_buffer[2] = static_cast<uint8_t>(channel_b_value_2 >> 8);
    output_buffer[3] = static_cast<uint8_t>(channel_b_value_2 & 0xFF);
    output_buffer[4] = static_cast<uint8_t>(channel_c_value_2 >> 8);
    output_buffer[5] = static_cast<uint8_t>(channel_c_value_2 & 0xFF);
    output_buffer[6] = static_cast<uint8_t>(channel_d_value_2 >> 8);
    output_buffer[7] = static_cast<uint8_t>(channel_d_value_2 & 0xFF);

    I2CHandle::Result i2cResult_3= _i2c.TransmitDma(0x64, &output_buffer[0], 8,
NULL, NULL);
    if(i2cResult_3 == I2CHandle::Result::OK) {
        seed.Println("OK TRANSMISSION 3");
    }
    I2CHandle::Result i2cResult_4= _i2c.TransmitDma(0x60, &output_buffer[0], 8,
NULL, NULL);
    if(i2cResult_4 == I2CHandle::Result::OK) {
        seed.Println("OK TRANSMISSION 4");
    }
}

```

```
}
```

ADC MUX - WORKING EXAMPLE USING 2 MUX AND MULTI-CHANNEL ADC CONFIG

```
/** Example of initializing multiple ADC inputs */
#include "daisy_seed.h"
/** This prevents us from having to type "daisy::" in front of a lot of things. */
using namespace daisy;
/** Global Hardware access */
DaisySeed hw;
int main(void)
{
    /** Initialize our hardware */
    hw.Init();
    /** Configure the ADC
     *
     * Three CV inputs (-5V to 5V -> 3V3 to 0V)
     * A0, A6, and A2
     *
     * This example was made for the Daisy Seed2 DFM, but the pins can be swapped for
     * other hardware.
     */
    AdcChannelConfig adc_cfg[2];
    adc_cfg[0].InitMux(seed::A0, 8, seed::D0, seed::D1, seed::D2);
    adc_cfg[1].InitMux(seed::A1, 8, seed::D0, seed::D1, seed::D2);

    /** Initialize the ADC with our configuration */
    hw.adc.Init(adc_cfg, 2);
    /** Start the ADC conversions in the background */
    hw.adc.Start();
    /** Startup the USB Serial port */
    hw.StartLog();
    /** Infinite Loop */
    while(1)
    {
        /** Print the values via Serial every 250ms
         * Values will be 0 when inputs are 0V
         * Values will be 65536 when inputs are 3v3
         *
         * Based on the CV input circuit this means that
         * Values will be 0 when input is 5V
         * Values will be ~32768 when input is ~0V
         * Values will be 65536 when input is -5V
         */
        System::Delay(50);
```

```

        float val_0 = hw.adc.GetMuxFloat(0, 0);
        float val_1 = hw.adc.GetMuxFloat(0, 1);
        float val_2 = hw.adc.GetMuxFloat(0, 2);
        float val_3 = hw.adc.GetMuxFloat(0, 3);
        float val_4 = hw.adc.GetMuxFloat(0, 4);
        float val_5 = hw.adc.GetMuxFloat(0, 5);
        float val_6 = hw.adc.GetMuxFloat(0, 6);
        float val_7 = hw.adc.GetMuxFloat(0, 7);
        float val_8 = hw.adc.GetMuxFloat(1, 0);
        float val_9 = hw.adc.GetMuxFloat(1, 1);
        float val_10 = hw.adc.GetMuxFloat(1, 2);
        float val_11 = hw.adc.GetMuxFloat(1, 3);
        float val_12 = hw.adc.GetMuxFloat(1, 4);
        float val_13 = hw.adc.GetMuxFloat(1, 5);
        float val_14 = hw.adc.GetMuxFloat(1, 6);
        float val_15 = hw.adc.GetMuxFloat(1, 7);
    //    float val_0 = hw.adc.GetFloat(0); //to get vals from 0 to 1
        hw.Print("      FLT_FMT3, FLT_VAR3(val_0));
        hw.Print("      FLT_FMT3, FLT_VAR3(val_1));
        hw.Print("      FLT_FMT3, FLT_VAR3(val_2));
        hw.Print("      FLT_FMT3, FLT_VAR3(val_3));
        hw.Print("      FLT_FMT3, FLT_VAR3(val_4));
        hw.Print("      FLT_FMT3, FLT_VAR3(val_5));
        hw.Print("      FLT_FMT3, FLT_VAR3(val_6));
        hw.Print("      FLT_FMT3, FLT_VAR3(val_7));
        hw.Print("*****");
        hw.Print("      FLT_FMT3, FLT_VAR3(val_8));
        hw.Print("      FLT_FMT3, FLT_VAR3(val_9));
        hw.Print("      FLT_FMT3, FLT_VAR3(val_10));
        hw.Print("      FLT_FMT3, FLT_VAR3(val_11));
        hw.Print("      FLT_FMT3, FLT_VAR3(val_12));
        hw.Print("      FLT_FMT3, FLT_VAR3(val_13));
        hw.Print("      FLT_FMT3, FLT_VAR3(val_14));
        hw.Print("      FLT_FMT3, FLT_VAR3(val_15));
        hw.Println(" ");
    }
}

```

SEED WORKING UART

Checkout branch seed-uart-test

CORRECT ORIENTATION FOR SLIDER POTENTIOMETER



Libre Auto-run

Auto Enable uarta overlay

1. sudo vim /etc/systemd/system/activate_uart_a.service
2. Write the following content in the file

3.

```
[Unit]
Description=Activate uart_a
Wants=network.target
After=network.target

[Service]
ExecStart= sudo ldto enable uart-a
WorkingDirectory=/home/$USER/GrooveTransformerPi/
User=$USER
Restart=always

[Install]
WantedBy=default.target
```

4. sudo systemctl enable activate_uart_a.service
5. sudo systemctl daemon-reload

1.

Autorun python script

1. sudo vim /etc/systemd/system/groove-transformer.service
2. Write the following content in the file
- 3.

```
[Unit]
Description=Groove Transformer Service
Wants=network.target
After=network.target

StartLimitBurst=5

[Service]
ExecStart=/usr/bin/python3 /home/$USER/GrooveTransformerPi/run_generative_engine.py
--baudrate 115200 --update_existing_state_after_AB_change 1 --model_name 100.pth
WorkingDirectory=/home/$USER/GrooveTransformerPi/
User=$USER
Restart=on-failure
```

```
StartLimitBurst=5

StandardOutput=/home/$USER/GrooveTransformerPi/systemctl_groove_transformer.out
StandardError=/home/$USER/GrooveTransformerPi/systemctl_groove_transformer.err

[Install]
WantedBy=default.target
```

4. sudo chmod +x /home/\$USER/GrooveTransformerPi/run_generative_engine.py
5. sudo systemctl enable /etc/systemd/system/groove_transformer.service

NOTE: If you change any of the systemctl files, you should reload service: **sudo systemctl daemon-reload**

Libre Setup

Connect Keyboard and Monitor

1. From <https://distro.libre.computer/ci/raspbian/11/> download raspbian lite ([2022-09-22-raspbian-bullseye-arm64-lite+aml-s905x-cc.img.xz](#))
2. Use Raspberry Pi Imager to flash to sd card
3. Login and setup
 - a. sudo raspi-config
 - b. Go to interface options, Enable SSH
4. Connect Ethernet and write ifconfig
 - a. Get the ip

Ssh and Packages

5. ssh \$USER@192.168.3.3
6. Install python
 - a. sudo apt update; sudo apt install python3 idle3; sudo apt install python3-pip;

7. Install Torch
 - a. pip install torch==1.13
8. sudo pip3 install pyserial

Github

9. sudo apt install git
10. <https://docs.github.com/en/authentication/connecting-to-github-with-ssh/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent?platform=linux>

Clone repo

11. git clone <git@github.com>:anon398/GrooveTransformerPi.git
12. git checkout remotes/origin/dev/SonarPrototypeDualInterpolation