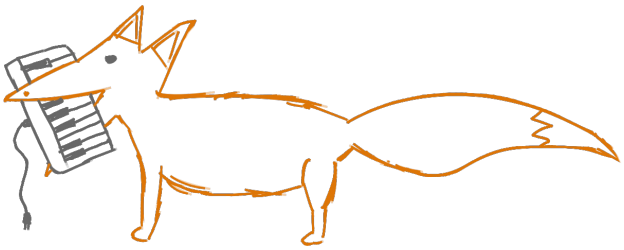
MidiFox

a Eurorack Module



The MidiFox is a small Eurorack Module.
It takes a MIDI-Signal via MicroUSB from a Computer and turns it into Eurorack usbale Note Value, MIDI Velocity and Gate CV.

Contents

Introduction

What is an eurorack Module anyway?

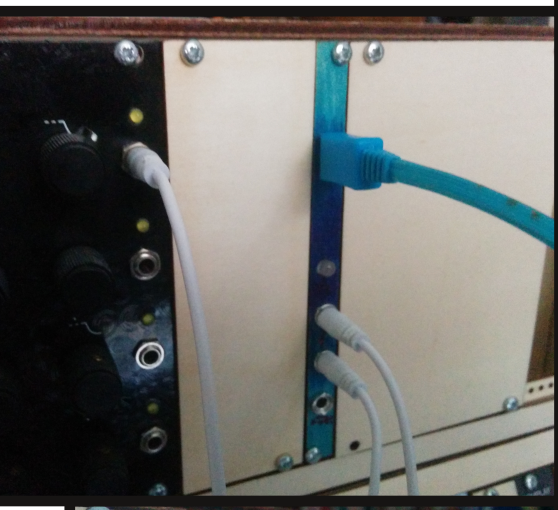
Eurorack is one standard for modular synthesizers. Modular synthesizers on the other hand are synthesizers (electronic instruments) that consist of many different modules (Things that do a small part of something).

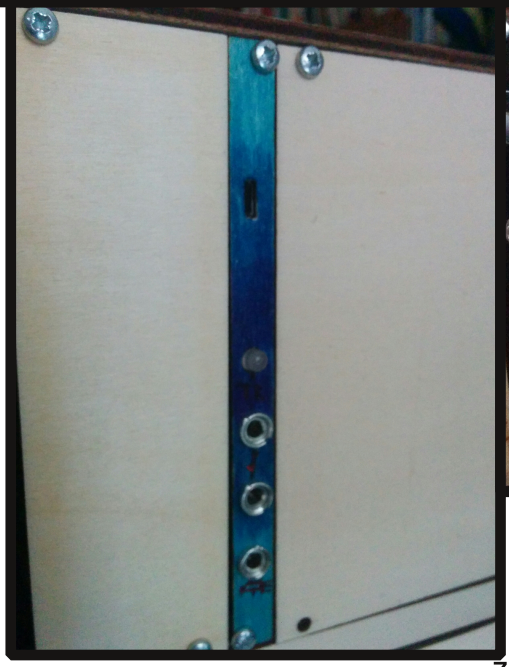
That means that in contrast to “classic” synthesizers where the signal path is predetermined by the manufacturer, in modular synthesis you can have your own signal path by chaining together modules in any way you want.

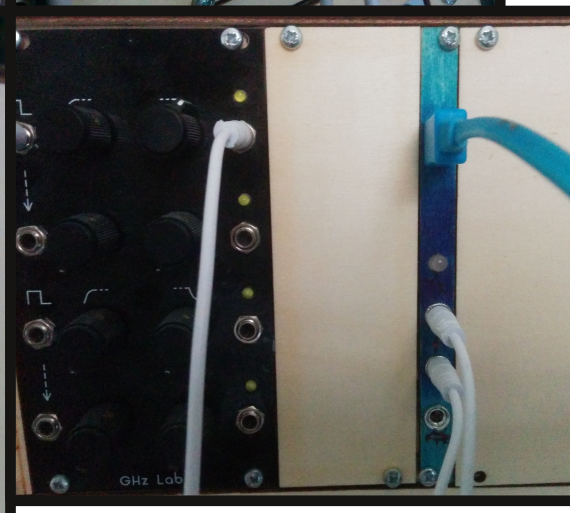
So whats the other stuff you talked about?

MIDI (short for: Musical Instrument Digital Interface) is a communication protocol that is used by electronic controllers (for example keyboards) to tell synthesizers what they have to synthesize. Commonly this is which note you pressed (corresponding to pitch information) how long your pressed it (corresponding to the so called gate) and how hard, or fast you pressed the key (corresponding to the MIDI Velocity.)

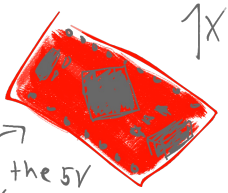
Photos







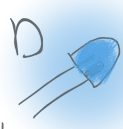
Bill of Materials + Tools



1x Sparkfun Pro Micro

get the 5V Variant

1x MCP4922 (Dual channel DAC)



LED (pick any color)

3x 3.5mm Audio Jack

1 Resistor (for the LED, select value accordingly)

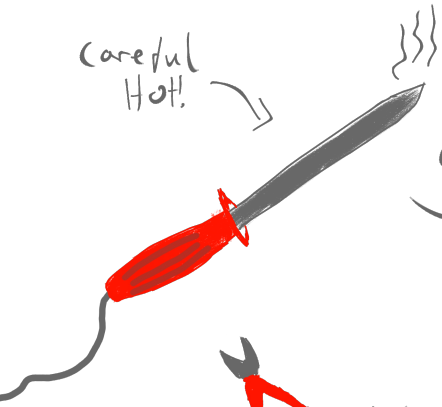
1x 100nF capacitor

Wire (some)

Computer (cat videos optional)

USB cabel

careful Hot!

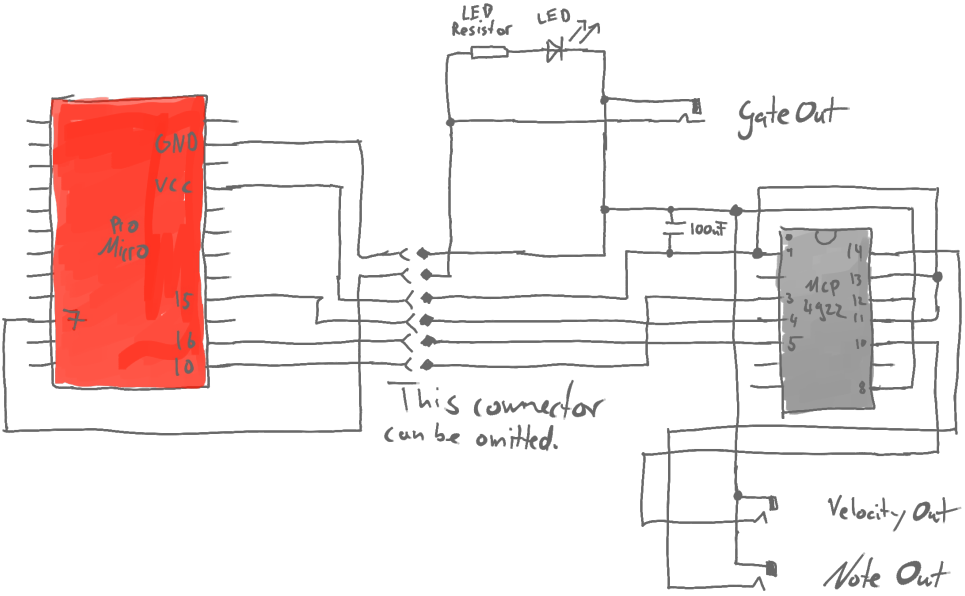


Soldering Iron

Solder

Wire Cutters

Schematic



The schematic for the MidiFox Eurorack Module.

Code Explanation

After set-up we wait in them main loop for a MIDI packet to arrive.

When one does we unpack it and put its content into three seperate bytes. One for the header, and one for each of the two data bytes. We care mainly for the MIDI Events on and off with the header 0x09 or 0x08 respectively.

First things first we set the gate, when we get a MIDI on message. The gate pin is connected to the gate ouput and gate LED, and here we need values of 0 or 5 Volts.
We do that via digitalWrite setting the pin to "HIGH". And since we are using a 5 Volt Microcontroller that initially unspecified amount means exactly the 5 Volts we needed.

Next we want to build the first set of bytes for our DAC. The first set of two bytes is for the note value. To get these values right, and to acess them fast we are using a so called "look-up table". A look-up table is a table full of precalculated values where can just "look-up" the right value. We then need to seperate this value into two 8 bit long bytes, for this we use the bitshift command (>>) to "move" the relevant bits 8 "digits" over. These two 8 bit integers then get passed as a Vector2, which is just a form of passing two values at once.

```
//read and split MIDI packet
  midiEventPacket_t event = MidiUSB.read(); //read Midi Signal to 'event'
  byte header = event.header;
  byte byteTwo = event.byte2;
  byte byteThree = event.byte3;



//if midi on
 if(header == 0x09){

   Serial.println("MIDI On");

   digitalWrite(GATE_OUT_PIN, HIGH);

   sendBytes(buildA(byteTwo));

   sendBytes(buildB(byteThree));

 }



/** Build byte A
 * Building byte a by use of the look-up table, than splitting the result
 * into two bits, which are passed in the form of a vector2 struct.
 */
vector2 buildA(byte noteByte){
  struct vector2 returnValue;
  returnValue.x = 80;
  int tableValue = noteTable[noteByte];
  returnValue.x += tableValue >> 8;
  returnValue.y = (byte) tableValue;
  return returnValue;
}
```

Now we send the data to our DAC. The DAC and Microcontroller comunicate via the Serial Peripheral Interface (SPI). The spefics of that standard do not concern us at this point, but the transmison works in practice like this: First we tell the Chip to pay atention, then we send our two bytes, and then lastely we tell the chip that we are done.

Now we are going to build our second set of bytes. This time this is even easier. On the incoming side we have the MIDI Velocity, which is per Defenition an integer value between 0 and 127. On the outgoing side we need a value between 0 and 5 Volts. But since the DAC primary function ist to convert the digital Values into analog ones from 0 up to 5 Volts, we only need to push the digits we got to the front. This is because the DAC works most significant bit first, meaning that the most important bites are the front ones. Otherwise we would only get differances in the second or third decimal place. We then split the value into two bytes similar to before, and send them exactly as before.

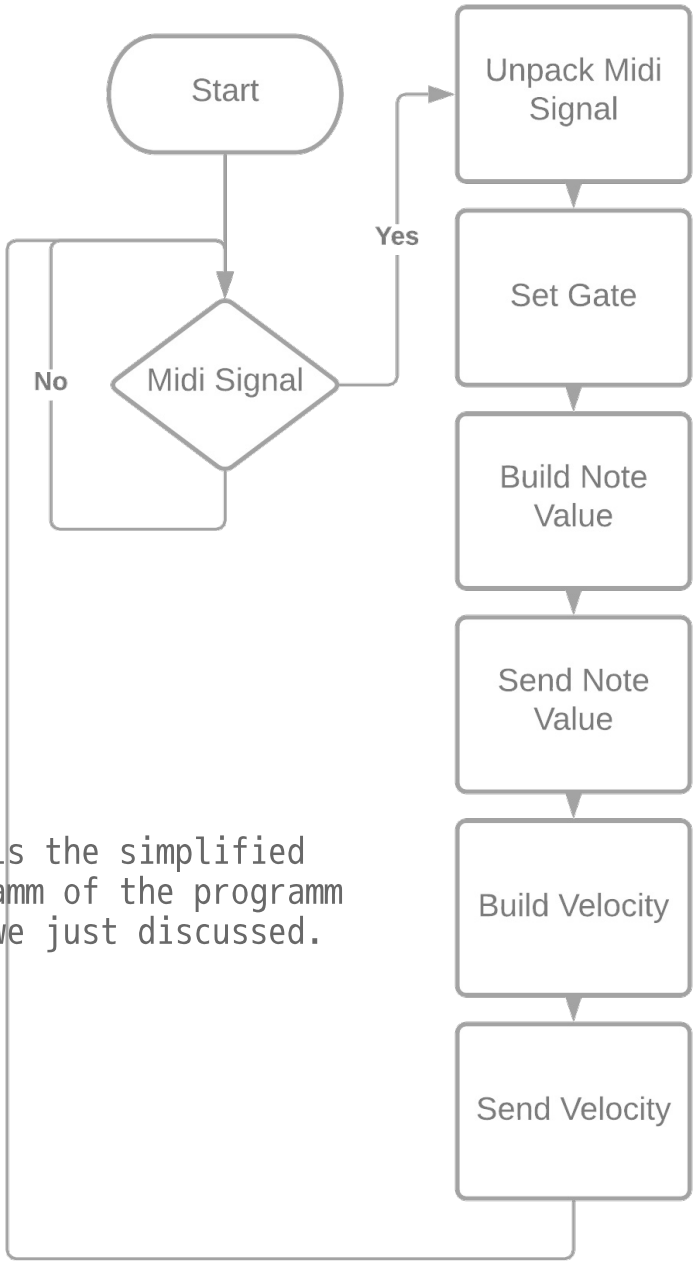
```
void sendBytes(vector2 data){

  digitalWrite(CHIP_SELECT_PIN, LOW); //activate DAC Communication

  SPI.transfer(data.x); //send first byte
  SPI.transfer(data.y); //send second byte

  digitalWrite(CHIP_SELECT_PIN, HIGH); //deactivate DAC Communication
}
```

```
/** Build byte B
 * Building byte a by multiplying, than splitting the result
 * into two bits, which are passed in the form of a vector2 struct.
 */
vector2 buildB(byte data){
  struct vector2 returnValue;
  returnValue.x = 208 + (data>>4);
  returnValue.y = data<<4;
  return returnValue;
}
```



This is the simplified diagramm of the programm flow we just discussed.

Attributions

Coded in Visual Studio Code
https://code.visualstudio.com/

Using platform.io for Arduino Codeing
https://platformio.org/

Schematic design in KiCad
https://kicad.org/

Frontpanel design in Fusion 360
https://www.autodesk.de/products/fusion-360/overview

DTP Layout in Scribus
https://www.scribus.net/

Drawings made in Krita
https://krita.org/en/

Digital font
http://mplus-fonts.osdn.jp/about-en.html#download-1

This Project can be found at
https://github.com/Felan7/midiFox