Erik Bartmann

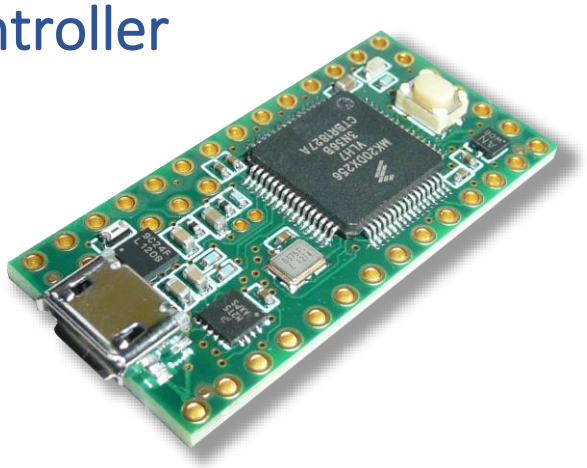
Modulare Synthesizer mit VCV-Rack 2 entdecken

Der Teensy MIDI-Controller

Autor	Erik Bartmann
Internet	https://erik-bartmann.de/
Thema	Der Teensy MIDI-Controller
Version	1.01
Datum	26. Januar 2022

Inhaltsverzeichnis

Ein Teensy-MIDI-Controller



Worum geht es überhaupt?

In diesem Papier werden folgende Themen besprochen.

- Was ist ein MIDI-Controller?
- Was ist ein Teensy-Board?
- Was ist MIDI?
- Der Teensy-MIDI-Controller
- Das Programm MIDIView
- Ein Test mit dem VCV-Rack

Wenn es darum geht, eine Kontrolle von Musiksoftware auf dem Computer über externe Hardware zu erlangen, dann kommen sogenannte MIDI-Controller ins Spiel. In diesem Papier geht es darum, einen rudimentären MIDI-Controller mit einem Mikrocontroller und einigen weiteren Bauteilen wie Drucktaster, Widerstände und Potentiometer zu entwickeln.

Das Teensy-Board

Auf dem Markt ist eine große Anzahl von Mikrocontrollern zu finden und gerade im Hobbybereich hat sich das Arduino-Board mit seinen unzähligen Varianten als Quasi-Standard durchgesetzt. Über die mitgelieferte Arduino-Entwicklungsumgebung (*Arduino-IDE*) wird es auch Neulingen ermöglicht, einen geeigneten Einstieg ohne eine steile Lernkurve zu finden. Doch das Board, worum es nun geht, ist nicht ein Arduino-Board, sondern das sogenannte *Teensy*-Board. Es passt perfekt auf ein Breadboard und ist in vieler Hinsicht leistungsfähiger, als ein Arduino-Standardboard wie zum Beispiel der Arduino-Uno. Das Teeny-Board wird von *Paul Stoffregen* entwickelt und ist optimal für den Bau eines USB-MIDI-Controllers geeignet, weil es sich sehr einfach als USB-MIDI-Gerät einsetzen und konfigurieren lässt.

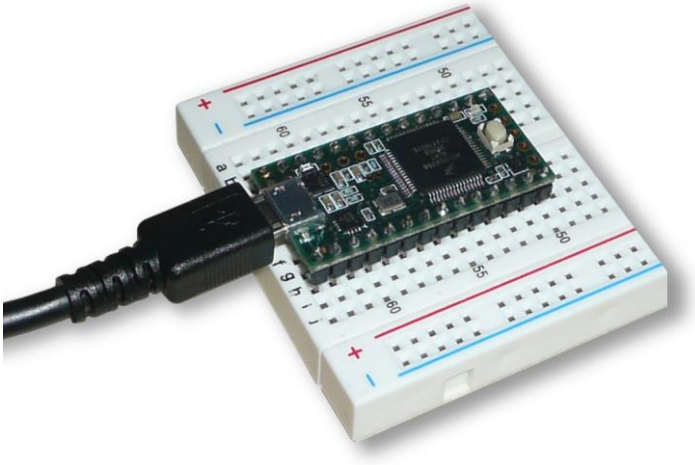


Abbildung 1 Der Teensy 3.2 auf einem Breadboard

Doch gehen wir die ganze Sache schrittweise an.

Die Installation der Entwicklungs-Software

Ich hatte gerade die Arduino-IDE kurz erwähnt, die einen sehr leichten Einstieg in die Programmierung von Arduino-Mikrocontrollern bietet. Es besteht jedoch noch ein weiterer und entscheidender Vorteil. Diese IDE kann über externe Software dahingehend erweitert werden, dass auch andere Mikrocontroller darüber programmiert werden können. So ist dies auch beim Teensy-Board der Fall. Doch zuvor muss natürlich die Arduino-IDE installiert werden.

Schritt 1: Die Installation der Arduino-IDE

Über den folgenden Link kann die Arduino-Entwicklungsumgebung für unterschiedliche Plattformen wie *Windows*, *Linux* und *MacOS X* heruntergeladen werden.

	Hyperlink!
https://www.arduino.cc/en/software	

Nach der einfachen Installation geht es nun im zweiten Schritt darum, die Teensy-Erweiterung zu installieren.

Schritt 2: Die Installation der Teensy-Erweiterung

Über den folgenden Link kann die Teensy-Installationssoftware heruntergeladen werden.

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

Nach der ebenfalls sehr einfachen Installation kann ein erster Test des Teensy-Boards durchgeführt werden.

Schritt 3: Der erste Test des Teensy-Boards

Natürlich muss das Teensy-Board über ein geeignetes USB-Kabel mit dem Computer verbunden werden. Nach der erfolgten Verbindung wird das Board automatisch erkannt, so dass kein zusätzlicher Treiber erforderlich ist. Die Installationssoftware der Teensy-Erweiterung für die Arduino-IDE besitzt alle erforderlichen Komponenten, um sofort mit der Programmierung starten zu können. Zu Beginn muss natürlich sichergestellt sein, dass das richtige Board in der Arduino-IDE ausgewählt ist. Da ich das *Teensy-Board 3.2* nutze, schaut das wie folgt aus. Unter dem Menüpunkt *Werkzeuge|Board* ist dieses Board auszuwählen.

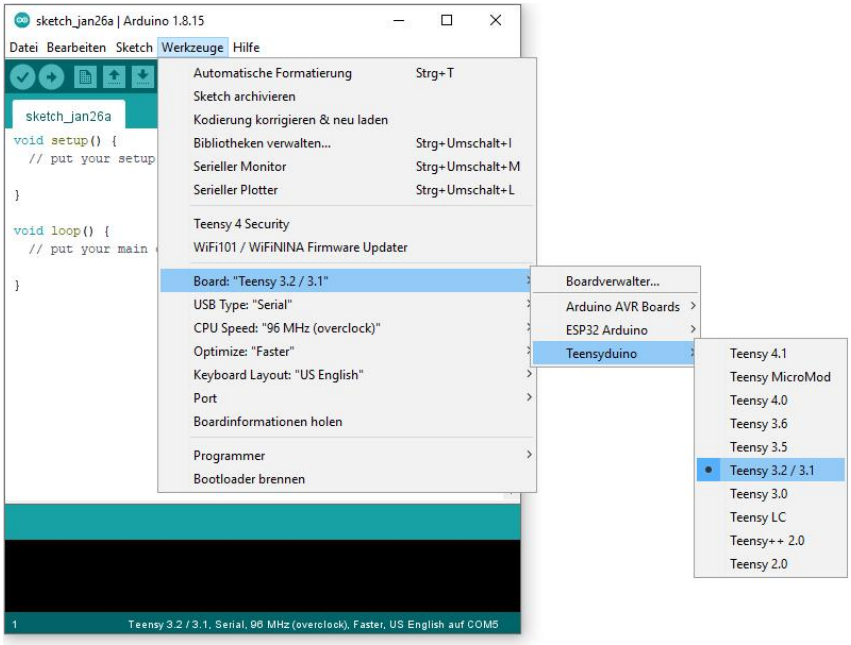


Abbildung 2 Das Teensy 3.2-Board ist ausgewählt

Im nächsten Schritt geht es um den korrekten Kommunikations-Port, der über den Menüpunkt *Werkzeuge|Port* auszuwählen ist. In der Regel ist das der einzige Port, der dort zur Auswahl angeboten wird und eine etwas kryptisch anmutet und vielleicht auch anders lauten kann, als hier von mir gezeigt.

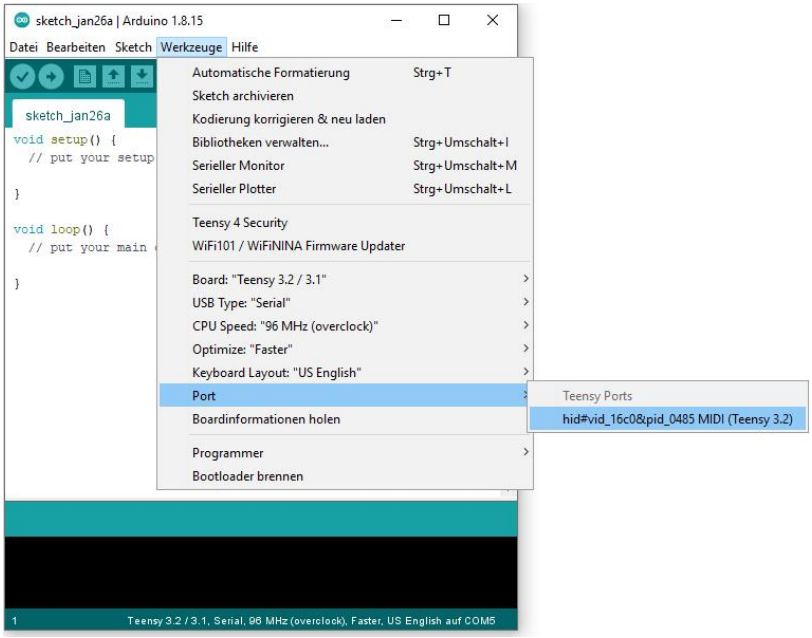


Abbildung 3 Der Teensy-Port ist ausgewählt

Im letzten Schritt kann dann für einen ersten Test der obligatorische Blink-Sketch geladen werden. Das erfolgt über den Menüpunkt *Datei|Beispiele|Teensy|Tutorial1|Blink*.

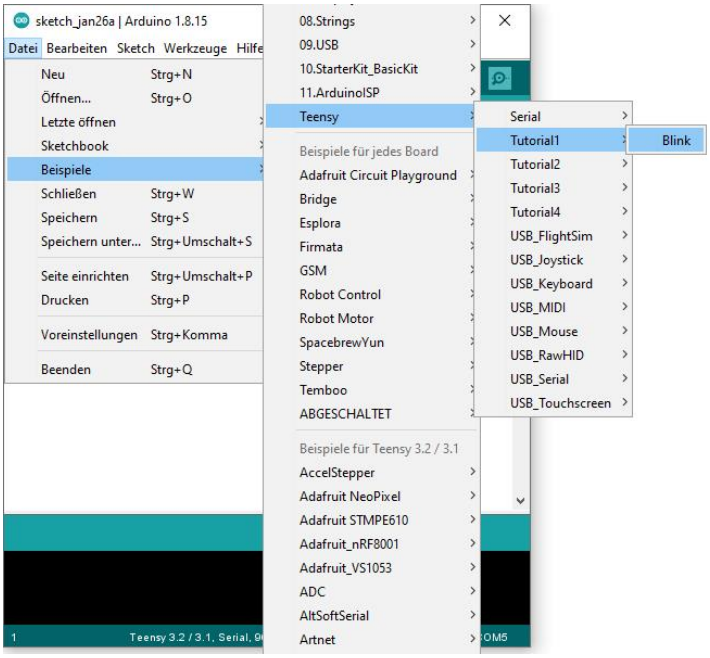


Abbildung 4 Das Laden des Blink-Sketches

Der Sketch schaut wie folgt aus.

```
const int ledPin = 13; // LED-Pin

void setup() {
  pinMode(ledPin, OUTPUT); // Output-Pin
}

void loop() {
  digitalWrite(ledPin, HIGH);   // LED on
  delay(1000);                  // 1 Sec. Pause (1000ms)
  digitalWrite(ledPin, LOW);    // LED off
  delay(1000);                  // 1 Sec. Pause (1000ms)
}
```

Nach dem Upload des Blink-Sketches meldet sich ein kleines Teensy-Hilfsprogramm, das einem bei dem Upload und dem Starten des Sketches hilft.

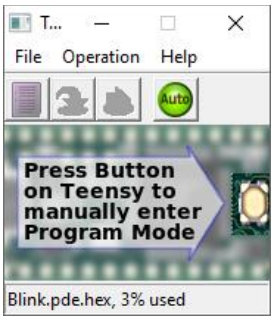


Abbildung 5 Das Teensy-Hilfsprogramm

Es kann erforderlich sein, dass man den kleinen weißen Drucktaster auf dem Board drücken muss, um einen Reset zu bewirken, was aber in der Regel nicht erforderlich ist. Nach dem erfolgten Upload sollte eine kleine Leuchtdiode (LED) auf dem Teensy-Board im Sekundentakt anfangen zu blinken. Wenn das der Fall ist, ist offensichtlich alles richtig gemacht worden und ein Zeichen dafür, dass die Kommunikation zwischen Computer und Mikrocontroller-Board bestens funktioniert.

Die Konfiguration als MIDI-Gerät

Das Teensy-Board soll als MIDI-Gerät bzw. -Device in Erscheinung treten und dafür ist eine spezielle Konfiguration erforderlich. Diese wird unter dem Menüpunkt *Werkzeuge|USB-Type|MIDI* ausgewählt.

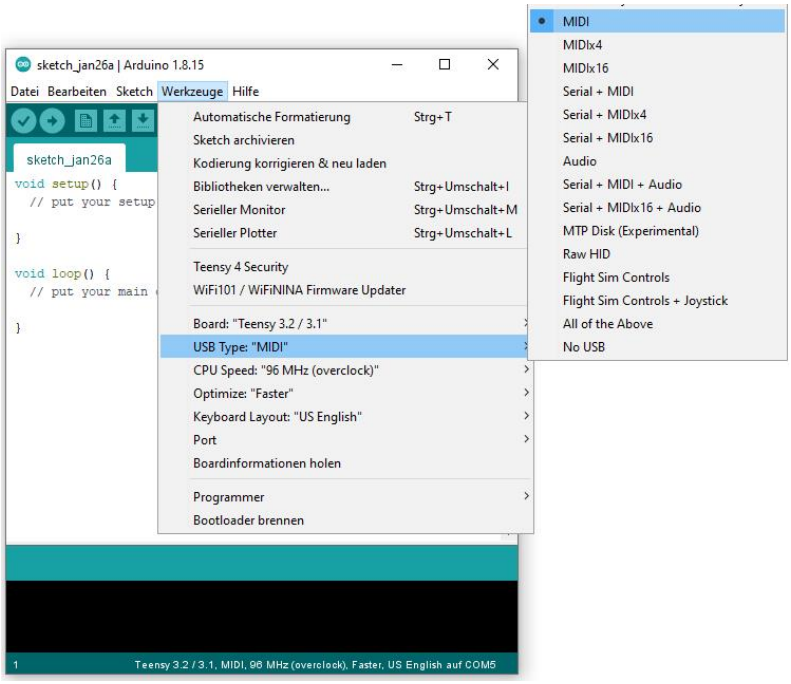


Abbildung 6 Der richtige USB-Type

Der Teensy als MIDI-Controller

Nun ist es an der Zeit, vom einfachen Blink-Sketch etwas in Richtung *MIDI* zu unternehmen. Bei MIDI handelt es sich um eine digitale Schnittstelle für Musikinstrumente und steht für ***M**usical **I**nstrument **D**igital **I**nterface*. Nähere Details dazu sind in einem speziellen Tutorial „*Was ist MIDI?*“ auf meiner Internetseite zu finden.

	Hyperlink!
https://erik-bartmann.de/userfiles/downloads/Musik/VCV-Rack/DE_MIDI.pdf https://erik-bartmann.de/userfiles/downloads/Musik/VCV-Rack/EN_MIDI.pdf	

Auch hier kann ein erster Test erfolgen, denn es sollte ja sichergestellt sein, dass auch alles in Richtigen Bahnen läuft. Dieser MIDI-Test besteht aus zwei Teilen, wobei der erste zeigen soll, wie die versendeten MIDI-Informationen auf unterster Ebene ausschauen und im zweiten Teil dann auch etwas zu hören ist.

Erster MIDI-Test

Für den ersten MIDI-Test muss zur Anzeige der MIDI-Informationen eine geeignete Software installiert werden, die frei verfügbar ist, sich *MIDIView* nennt und unter dem folgenden Link zu bekommen ist.

Hyperlink!

https://hautetechnique.com/midi/midiview/

Es handelt sich um ein sogenanntes Monitor-Programm, das alles versendeten MIDI-Daten zur Anzeige bringt. Doch dazu später gleich mehr. Zuerst muss natürlich ein passender Teensy-Sketch programmiert werden, damit auch etwas zu sehen und später zu hören ist. Das Teensy-Board soll jetzt Standard-MIDI-Nachrichten versenden. Hört sich schon etwas merkwürdig an, oder?! Ganz einfach! Eine derartige Nachricht kann zum Beispiel so aussehen, dass eine Note gespielt und dann wieder verklingen soll. Das nennt sich im Detail

- Note on
- Note off

Derartige Ereignisse, auch Events genannt, werden natürlich in der Teensy-MIDI-Programmierung unterstützt. Sie lauten

- `usbMIDI.sendNoteOn(note, velocity, channel);`
- `usbMIDI.sendNoteOff(note, velocity, channel);`

Die drei Argumente beim Aufruf einer derartigen Methode (Methoden sind eigentlich Funktionen und werden in der *Objektorientierten Programmierung* so genannt) haben die folgende Aufgabe.

- *note*: Tonhöhe - auch Pitch genannt
- *velocity*: Anschlagstärke
- *channel*: der verwendete MIDI-Kanal

Nun kann man den verwendeten Blink-Sketch sehr einfach um diese beiden Methoden erweitern, um dann eine Note zu spielen und dann wieder nicht.

```
const int ledPin = 13; // LED-Pin

void setup() {
  pinMode(ledPin, OUTPUT); // Output-Pin
}

void loop() {
  digitalWrite(ledPin, HIGH);      // LED on
  usbMIDI.sendNoteOn(48, 99, 1);   // Note on (C2 / C3)
  delay(1000);                     // 1 Sec. Pause
  digitalWrite(ledPin, LOW);       // LED off
  usbMIDI.sendNoteOff(48, 0, 1);   // Note off (C2 / C3)
  delay(1000);                     // 1 Sec. Pause
}
```

Es ist zu erkennen, dass beim Stummschalten einer Note einfach der Velocity-Wert 0 genutzt wird, was so viel bedeutet, dass die Taste auf einem Keyboard überhaupt nicht angeschlagen wird. Es hätte also auch die folgende Zeile verwendet werden können.

```
...
  usbMIDI.sendNoteOn(48, 0, 1);
...
```

Der Sketch reagiert wie vorher, dass weiterhin die kleine LED im Sekundentakt blinkt, doch nun werden auch in diesem Intervall die zwei Nachrichten *Note on* und *Note off* über die genannten Methoden versendet. Ein Blick in das Programm *MIDIView* zeigt die folgenden fortlaufenden Nachrichten.

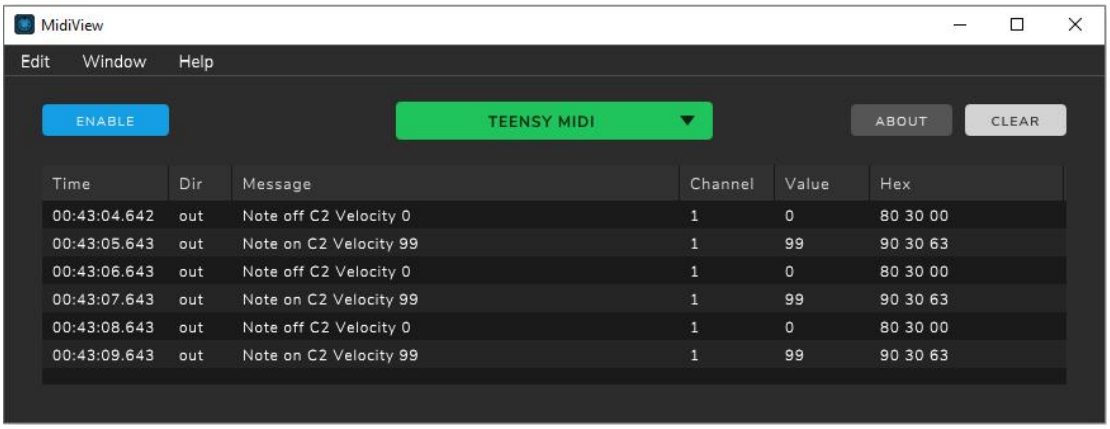


Abbildung 7 Die MIDI-Nachrichten in MIDIView

In der Nachrichtenspalte, die mit *Messages* gekennzeichnet ist, kann man sehen, dass die Note mit der Tonhöhe *C2* erkannt wurde. Im entsprechenden Teensy-Sketch habe ich als Kommentar jedoch *C2* beziehungsweise *C3* geschrieben. Warum dieser Unterschied um eine Oktave? Das liegt darin begründet, dass es unterschiedliche Standards gibt, was ja eigentlich dem Standard widerspricht ein Standard zu sein. Verwirrend, oder?! Es gibt zwei Standards, wobei das sogenannte *Middle-C* bei *Yamaha* bei C3 und bei *Roland* bei C4 liegt. Der im Sketch verwendete MIDI-Notenwert, der mit 48 angegeben ist, repräsentiert also den Tonwert C2 mit einer Frequenz von 130,81Hz. Ich habe dazu einen VCV-Rack-Patch vorbereitet, der das *HOT TUNA*-Plugin von *NYSTHI* nutzt, um darin sowohl die Frequenz, als auch die erkannte Note anzuzeigen. Ich komme aber noch im Detail auf diesen Patch zu sprechen.

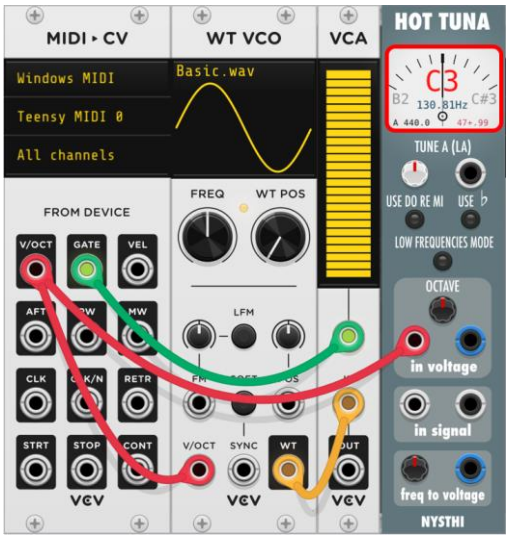


Abbildung 8 Der VCV-Rack-Patch zur Anzeige von Note und Frequenz

Ein kleiner MIDI-Controller

Nun kommen wir zu etwas, was wirklich im realen Betrieb gebracht werden kann. Es geht um einen sehr einfachen MIDI-Controller, der mit vier Drucktasten und zwei Drehreglern ausgestattet ist. Dieser kann natürlich nach Belieben erweitert werden. Ich denke, dass dieses kleine Projekt Lust auf mehr macht. Lassen wir uns beginnen. Hierzu ein bisschen Theorie, was detailliert in dem Tutorial „*Was ist MIDI?*“ auf meiner Internetseite zu lesen ist. Es gibt in MIDI acht unterschiedliche Kategorien beziehungsweise *Befehlstypen*, wie das in der folgenden Tabelle zu sehen ist.

Befehlstyp	Status-Byte (binary)	Status (hex)
Note off	1000 *nnnn*	8*n*
Note on	1001 *nnnn*	9*n*
Poly Pressure	1010 *nnnn*	A*n*
Control Change	1011 *nnnn*	B*n*
Program Change	1100 *nnnn*	C*n*
Channel Pressure	1101 *nnnn*	D*n*
Pitch-Bend	1110 *nnnn*	E*n*

Tabelle 1 MIDI-Befehlstypen

Die beiden ersten wurden schon beim ersten MIDI-Test verwendet, worüber Noten versendet werden. Nun gibt es aber noch den Befehlstyp *Control Change* (CC), der dafür da ist, etwaige Parameter eines Synthesizers zu modifizieren (Modulation). Das kann zum Beispiel dann der Fall sein, wenn die Tonhöhe (Pitch) oder die Cut-Off-Frequenz verändert werden. Das wird dann in der Regel über Drehregler gemacht, die jetzt ins Spiel kommen. Der geplante MIDI-Controller soll also Tasten (Buttons) zur Steuerung der Tonhöhe und Drehregler (Pots) zu Anpassung verschiedener Parameter besitzen, wie das auf der folgenden Abbildung schematisch zu sehen ist.

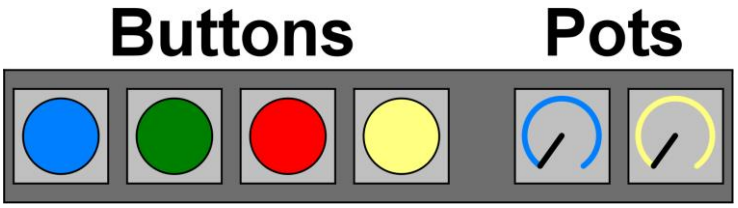


Abbildung 9 Die Bedienelemente des MIDI-Controllers

Wird einer der vier Tasten gedrückt und wieder losgelassen, soll ein entsprechender *Note on*- bzw. *Note off*-Befehl mit einer bestimmten Tonhöhe und einem Velocity-Wert versendet werden. Bei den Drehreglern sieht es ähnlich aus. Dort soll die jeweilige Position ermittelt werden, um diese dann zur Manipulation eines gewünschten Parameters zu nutzen. Damit nicht kontinuierlich MIDI-Informationen versendet werden, kommt es im Endeffekt nur dann dazu, wenn sich wirklich ein Wert ändert. Gerade bei den Potentiometern ist das ein Problem, auf das ich noch zu sprechen komme. Hinsichtlich eines Control-Change muss erwähnt werden, dass dieser in einem MIDI-Controller natürlich identifiziert werden muss. Es sollte ja klar sein, welcher Drehregler überhaupt bewegt wurde. Aus diesem Grund werden diesen Elementen bestimmte *CC-IDs* zugewiesen. Für unser Vorhaben sind das die folgenden Werte.

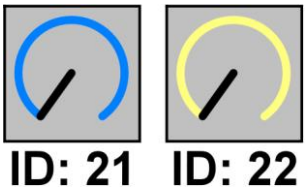


Abbildung 10 Die CC-IDs der beiden Potentiometer

Die genannten Controller-Elemente, also Drucktaster und Drehregler müssen jetzt in irgendeiner Weise mit dem Teensy-Board verbunden werden. Dazu sollte vorher klar sein, welche einzelnen Anschlüsse auf dem Board, über welche Funktionen verfügen.

Das Pin-Layout des Teensy 3.2

Sehen wir uns nun das sogenannte Pinout des Teensy 3.2 etwas genauer an, wobei ich mich mehr oder weniger auf die für das Projekt erforderlichen Pin-Gruppen beschränke.

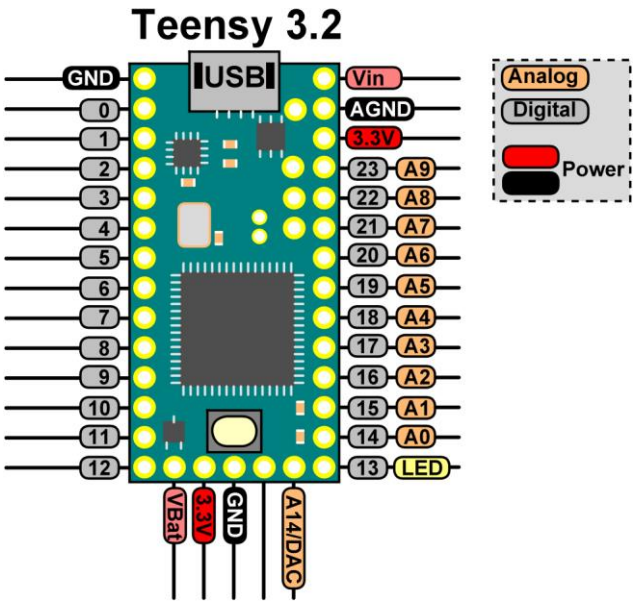


Abbildung 11 Das vereinfachte Pin-Out des Teensy 3.2

Es ist zu sehen, dass im Grund genommen zwei unterschiedliche Pin-Gruppen existieren. Zum einen die analogen Pins für das Messen kontinuierlicher Spannungswerte, die über die Drehregler geliefert werden und die digitalen Pins zur Abfrage, ob ein Taster gedrückt oder nicht gedrückt ist.

Der Schaltplan

Für die beiden Drehregler werde ich die analogen Pins *A0* und *A1* verwenden und die Taster mit den digitalen Pins *0*, *1*, *2* und *3* verbinden. Sehen wir uns das in einem Schaltbild genauer an, der alle elektrischen Verbindungen aufzeigt.

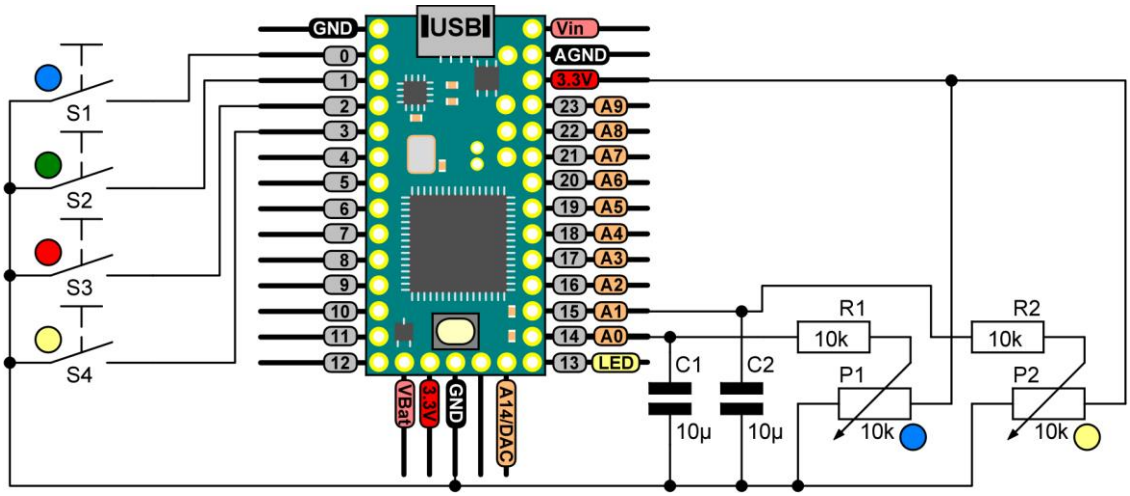


Abbildung 12 Der Schaltplan des MIDI-Controllers

Ich möchte nun ein Problem ansprechen, das ich schon kurz hinsichtlich der Drehregler angesprochen hatte. Es sollen ja nur *dann* MIDI-Informationen versendet werden, wenn sich ein Wert ändert. Angenommen, es liegt an einem Potentiometer ein Widerstandswert von 500Ω vor. Dieser wird dann von dem Programm in einen entsprechenden Spannungswert umgerechnet und weiterverarbeitet. Eine korrespondierende MIDI-Information wird versendet. Ändert sich dieser Wert nicht, weil der Drehregler nicht mehr in seiner Position verändert wird, soll auch kein weiterer MIDI-Versand stattfinden. Nun hat aber ein Potentiometer die Eigenschaft, dass er aufgrund von mechanischen Eigenschaften oder kleinen Verunreinigungen, vielleicht an der aktuellen Position immer zwischen zwei Widerstandswerten hin und herschwankt, was *Jitter* genannt wird. Die fortlaufenden Messergebnisse sind also zum Beispiel 500Ω, 501Ω, 500Ω, 499Ω, 500Ω, usw. Das führt natürlich dazu, dass das Programm denkt, der Drehregler würde ständig bewegt und sendet daraufhin kontinuierlich nicht gewünschte MIDI-Informationen. Nun kann man das programmtechnisch versuchen in den Griff zu bekommen. Zusätzlich rate ich jedoch noch zu einer kleinen Erweiterung in Form eines sogenannten Tiefpass-Filters, der auch in dem Schaltplan zu sehen ist und durch die zusätzlichen Widerstände und Kondensatoren realisiert wurde. Hier der Schaltplan eines Potentiometers mit nachgeschaltetem Tiefpass-Filter.

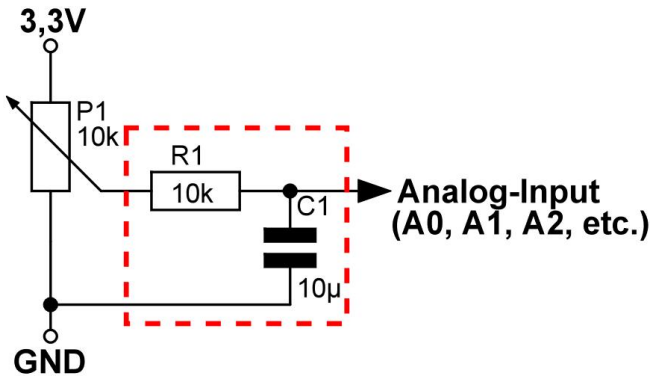


Abbildung 13 Der Tiefpass-Filter - Rot umrandet

Dieser Filter bewirkt bei sehr schnellen Änderungen eine Ableitung des Signals nach Masse (GND), so dass diese nicht zum analogen Eingang weitergeleitet werden und lässt eben nur niedrige Frequenzen passieren. Die mechanischen Taster besitzen ebenfalls einen unangenehmen Nebeneffekt. Diese können beim Schließen und Öffnen des Kontaktes kurzzeitig mehrere logische Pegel hintereinander anstatt nur einen abgeben. Dieses Verhalten wird *Prellen* (*Bouncing*) genannt und durch eine entsprechende Bibliothek kompensiert. Sehen wir uns jetzt einmal den Aufbau auf einem Breadboard an.

Der Aufbau auf dem Breadboard

Mit den genannten Bauteilen und der übersichtlichen Verkabelung ist der Aufbau auf einem Breadboard sehr schnell umgesetzt.

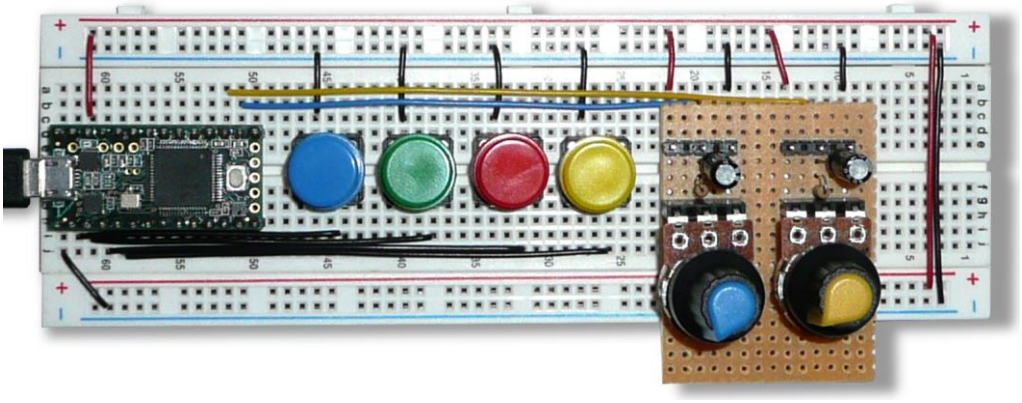


Abbildung 14 Der MIDI-Controller auf einem Breadboard

Die erforderlichen Bauteile sind.

- Teensy 3.2
- Breadboard 1x
- Mikrotaster 4x
- Potentiometer 10K linear 2x
- Widerstände 10K 2x
- Kondensatoren 10µ 2x
- Platinenrest + Stiftleisten (nur falls gewünscht)
- Kabel

Der Teensy-Sketch

Der Ausgangspunkt zur Realisierung ist ein Beispiel-Sketch, der sich unter dem folgenden Menüpunkt der Arduino-IDE verbirgt.

Datei|Beispiele|Teensy|USB_MIDI|Many_Button_Knobs

Dieser Sketch wurde von mir so modifiziert, dass er hinsichtlich *4 Buttons* (Taster) und *2 Knobs* (Drehregler) funktioniert. Ich zeige den Sketch hier nicht in seiner ganzen Länge, sondern nur die Anpassungen.

Festlegung der Anzahl der analogen und digitalen Pins:

```
...
const int A_PINS = 2; // number of Analog PINS
const int D_PINS = 4; // number of Digital PINS
...
```

Festlegung der analogen Eingangs-Pins und der Control-IDs der Potentiometer:

```
...
const int ANALOG_PINS[A_PINS] = {A0, A1}; // analog Input-Pins
const int CCID[A_PINS] = {21, 22};        // Control-ID
...
```

Festlegung der digitalen Eingangs-Pins und der MIDI-Notenwerte:

```
...
const int DIGITAL_PINS[D_PINS] = {0, 1, 2, 3}; // digital Input-Pins
const int note[D_PINS] = {60, 61, 62, 63};     // MIDI-Note-Values
...
```

Festlegung der analogen Eingangs-Pins hinsichtlich der Minimierung des Jitter-Effektes:

```
...
ResponsiveAnalogRead analog[]{
  {ANALOG_PINS[0],true},
  {ANALOG_PINS[1],true},
  {ANALOG_PINS[2],true},
  {ANALOG_PINS[3],true}
};
...
```

Festlegung der digitalen Eingangs-Pins zur Entprellung der Tasten:

```
...
Bounce digital[] =   {
   Bounce(DIGITAL_PINS[0],BOUNCE_TIME),
   Bounce(DIGITAL_PINS[1], BOUNCE_TIME),
   Bounce(DIGITAL_PINS[2], BOUNCE_TIME),
   Bounce(DIGITAL_PINS[3], BOUNCE_TIME)
};
...
```

Durch diese Änderungen ist der Sketch nun auf die verwendete Hardware und Verkabelung abgestimmt. Nach dem Upload des Sketches kann ein erster MIDI-Controller-Test mithilfe des *MIDIView*-Programms durchgeführt werden.

Das Testen des MIDI-Controllers

Nach dem Start von MIDIView und das Wahl des Teensy-MIDI-Controllers mit dem Namen *TEENSY MIDI* kann der Test gestartet werden. Zu Beginn sollte das Nachrichtenfenster leer sein und es sollten auch keine MIDI-Informationen dort auflaufen.

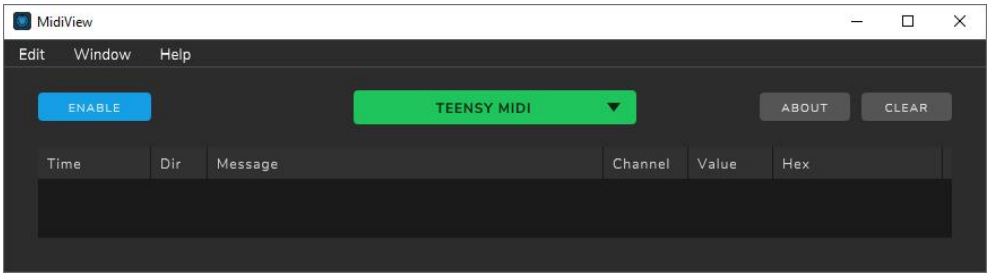


Abbildung 15 Ein leeres MIDIView-Fenster

Nun werde ich nacheinander die Tasten von links nach recht drücken und sehen, was sich in MIDIView so ereignet.

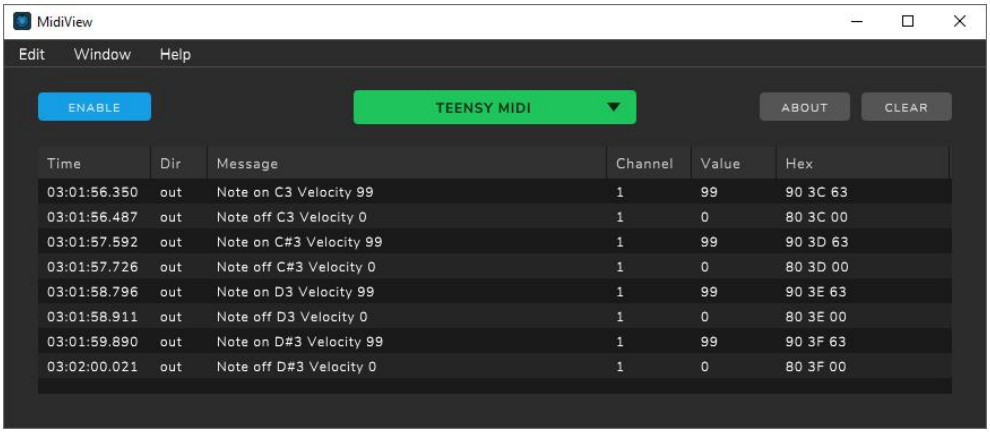


Abbildung 16 Die MIDI-Nachrichten der 4 Taster

Es ist wunderbar zu erkennen, dass es pro Taster sowohl ein *Note on*, als auch *Note off* Ereignis gibt und die im Sketch definierten MIDI-Notenwerte hier interpretiert werden. Das waren ja über die folgende Zeile definierten Werte.

```
...
const int note[D_PINS] = {60, 61, 62, 63};      // MIDI-Note-Values
...
```

Die folgende Tabelle zeigt die MIDI-Werte und die entsprechenden Noten, wobei ich die verwendeten MIDI-Werte in Rot markiert habe.

Oktave	Noten											
	C	C#	D	D#	E	F	F#	G	G#	A	A#	H (B)
-2	0	1	2	3	4	5	6	7	8	9	10	11
-1	12	13	14	15	16	17	18	19	20	21	22	23
0	24	25	26	27	28	29	30	31	32	33	34	35
1	26	37	38	39	40	41	42	43	44	45	46	47
2	48	49	50	51	52	53	54	55	56	57	58	59
3	60	61	62	63	64	65	66	67	68	69	70	71
4	72	73	74	75	76	77	78	79	80	81	82	83
5	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107
7	108	109	110	111	112	113	114	115	116	117	118	119
8	120	121	122	123	124	125	126	127				

Tabelle 2 MIDI-Noten-Werte

Es ist zu erkennen, dass es sich um die Noten *C*, *C#*, *D* und *D#* in der 3. Oktave handelt, was auch im MIDIView zu sehen ist. Dann wollen wir mal sehen, wie sich die beiden Drehregler zu erkennen geben. Ich bewege zuerst den linken ein wenig nach rechts, der sich zu Beginn im Linksanschlag befand.

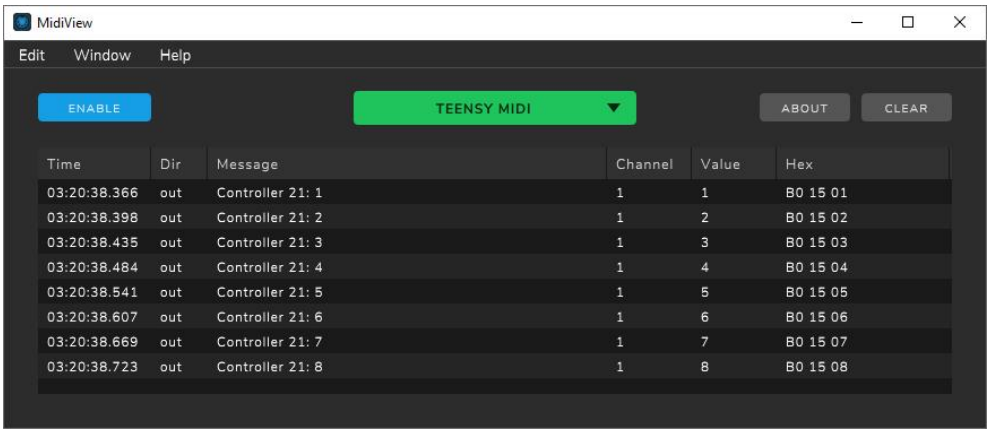


Abbildung 17 Die MIDI-Nachrichten des linken Drehreglers

Die beiden Drehregler wurden ja hinsichtlich ihrer CC-IDs wie folgt definiert.

```
...
const int CCID[A_PINS] = {21, 22};        // Control-ID
...
```

Und genau diese *ID 21* ist auch im MIDIView in der Spalte *Messages* zu erkennen. Zusätzlich wird natürlich noch der erkannte Wert zur Anzeige gebracht, der sich im Bereich von 0 bis 127 bewegen kann. Ich werde das nun mit dem rechten Drehregler machen, der sich zu Beginn im Rechtsanschlag befand und jetzt langsam nach links gedreht wird.

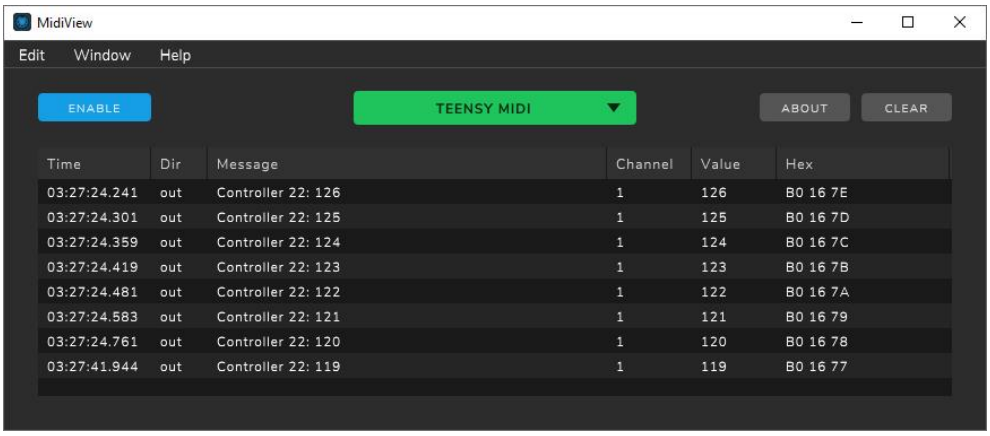


Abbildung 18 Die MIDI-Nachrichten des rechten Drehreglers

Die *ID 22* wird im MIDIView mit dem entsprechenden Wert zu Anzeige gebracht. Alles läuft also genau so, wie es zuvor im Sketch definiert wurde.

Eine andere MIDI-Kennung vergeben

Im Programm MIDIView war zu sehen, dass sich unser MIDI-Controller mit dem Namen *TEENSY MIDI* zu erkennen gibt. Das ist eine vom Sketch vordefinierte Kennung, die jedoch auf Wunsch sehr leicht angepasst werden kann. Ein entsprechendes Beispiel ist unter dem folgenden Menüpunkt zu finden.

Datei|Beispiele|Teensy|USB_MIDI|MIDI_name

Die Vorgehensweise ist die folgende.

Schritt 1: Einen neuen Tab-Reiter mit dem Namen name.c dem Sketch hinzufügen.

Schritt 2: Den folgenden Code einfügen.

```
#include "usb_names.h"

#define MIDI_NAME   {'E','r','i','k','\'','s',' ','M','I','D','I'}
#define MIDI_NAME_LEN  11

// Do not change this part.  This exact format is required by USB.
struct usb_string_descriptor_struct usb_string_product_name = {
        2 + MIDI_NAME_LEN * 2,
        3,
        MIDI_NAME
};
```

Der von mir gewünschte Name soll *Erik's MIDI* lauten und wurde über das MIDI_NAME-Array in Form von Einzelbuchstaben definiert. Die Länge des Arrays muss in der nächsten Zeile natürlich noch angepasst werden. Also einfach die Anzahl der Buchstaben zählen und dort als Wert hinterlegen. Nach dem Upload erscheint dann in MIDIView dieser neue Name zur Auswahl.

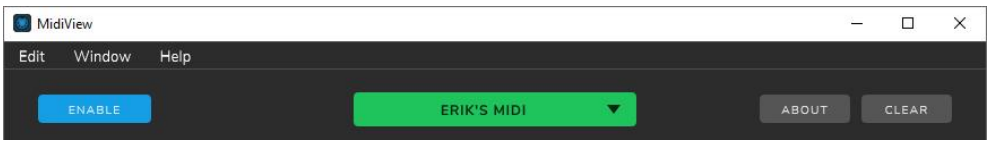


Abbildung 19 Der neue MIDI-Name

Ein richtiger Test in VCV-Rack

Bei so viel Theorie ist es sicherlich angebracht, den MIDI-Controller in einer richtigen Umgebung zu testen. Da bietet sich natürlich wieder das VCV-Rack an. Ich habe den folgenden Patch dafür vorbereitet.

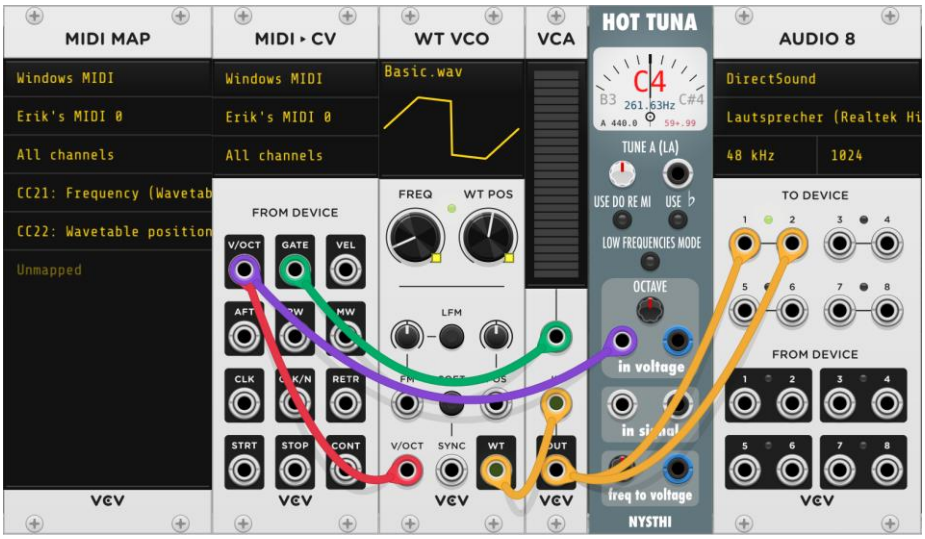


Abbildung 20 Der VCV-Rack-Patch für den Teensy-MIDI-Controller

Welche VCV-Rack-Module sind hierzu notwendig? Nun, das ist auf jeden Fall das linke Modul mit dem Namen *MIDI-MAP*. Dort kommt es zu einer Zuordnung eines MIDI-Controller-Bedienelementes zu einem VCV-Rack-Bedienelement, was durch das sogenannte *Mapping* erfolgt. Ich habe dazu ein spezielles Video erstellt. Über die beiden Drehregler am MIDI-Controller ist es jetzt möglich, die beiden Drehregler *FREQ* und *WT POS* am *WT VCO*-Modul zu beeinflussen.

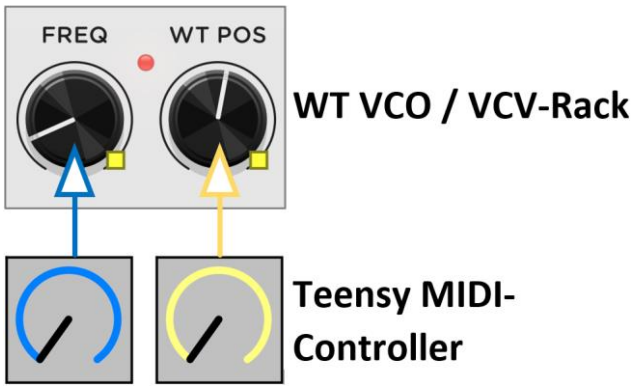


Abbildung 21 Die Drehregler beeinflussen das WT-VCO-Modul

Natürlich kann auch über die vier Tasten die Tonhöhe in sehr engen Grenzen entsprechend geändert werden. Dieses Beispiel dient ja lediglich als kleiner Einstieg in die Thematik der MIDI-Controller und ich hoffe, dass es dazu beiträgt, eine inspirierende Wirkung zu erzielen.

Weitere Informationen sind auf meiner Internetseite zu finden.

Hyperlinks!

https://erik-bartmann.de/
https://erik-bartmann.de/?Musik___VCV-Rack

Frohes Frickeln!

Erik Bartmann