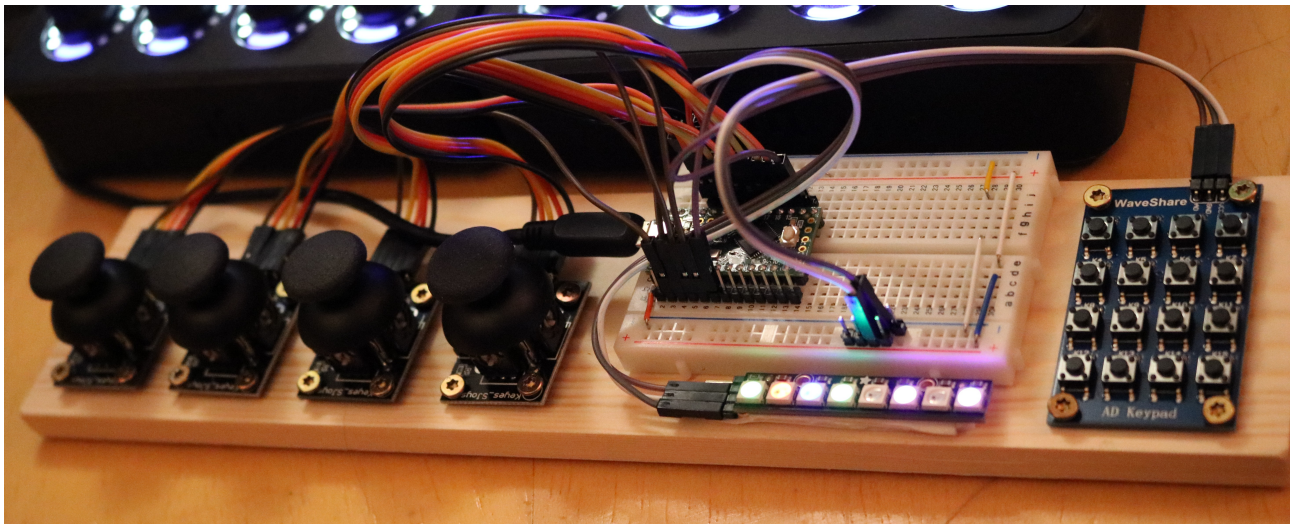
Motivation:

When trying to play an electronic instrument as expressive as you would an acoustic instrument, you're meeting the challenge to find and use the correct hardware and/or software to do just that in a meaningful way.

For instance there are specialized woodwind- or guitar-controllers, which are dependent on a suitable hard- or software backend-system to work together as a playable instrument. Until more recently it was almost impossible to find a really expressive, yet affordable solution for that.

So called MPE-capable controllers now fill that gap, one of the less expensive versions is a keyboard-based variant with 2 octaves, yet they are still not exactly cheap. (Starting at about $300).
That's what made me think about developing a cheaper DIY solution.

To give you an idea, here is a picture of the first prototype I came up with, using parts I already had lying around:



Disclaimer:
All product and company names are trademarks™ or registered ® trademarks of their respective holders. Use of them does not imply any affiliation with or endorsement by them.

Concept:

The basic concept of this DIY-controller is that you can modify the note played, very much as you could bend and/or modify the sound of each string on a guitar separately.
So this would be per-note modifiers, whereas with a standard synth, one typically would have a pitch-bend and modulation-wheel to modify all notes currently played at once.

The challenge to make this happen is not only using suitable hardware, but also to deliver the appropriate data to the actual synth-engine.
Because MIDI (Musical Instruments Digital Interface) was designed very much with keyboards in mind, this required some thought. Actually in the past with Midi-Guitars the same problems already got solved by using a separate MIDI-channel for each string. This way, for instance providing the per note pitch-bend was no problem, because each channel would have its own sound-generator being controlled by the individual bending-information.

Right now, more and more MPE-capable synths would pop up, especially as quite inexpensive virtual instruments, playable via your favourite DAW on your computer.

The main idea behind MPE now is to standardize the way this information can be provided, mapped and transported. MPE stands for MIDI Polyphonic Expression, sometimes also the term used especially with keyboards would be Multidimensional Polyphonic Expression.

MPE is not exactly a protocol-extension, but a convention of how already defined events can be used to possibly allow modifications for each and every note played.
In a nutshell MPE typically allows note-on-velocity, note-off-velocity, per note pitch-bend, per note brightness (Midi Controller information CC74) and per note “Aftertouch” (Pressure applied after hitting the note).

So it was apparent, that the DIY-controller should should support the MPE data-conventions, but maybe be capable to send data necessary for non-MPE-synths as well.

Design Decisions

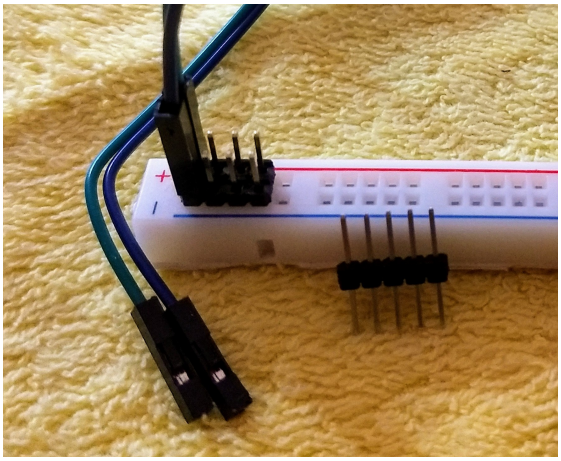
It turned out, that after some basic “design decisions” had been taken, building a quite durable MPE-controller could be done in a few hours with simple equipment.

The joysticks, very much as with the first prototype pictured at the beginning can be screwed on the board to be seen in the picture. It’s seven cm deep and one cm thick.

To fit in the microcontroller into a relatively small enclosure, a kind of pedestal has been chosen. Because the wooden border is 2 cm in height the microcontroller easily can be fitted into such a box. Also it would be possible to drill holes as outlets for wires at the back, that’s why the board is less thick there.

In addition to the microcontroller such a case also would provide enough space for all the cables needed, including the USB-connector and some kind of power-supply for the hardware to be on top of the pedestal.

More on these topics later-on.



Some information on MPE

If you do a web-search on MPE-controller or MPE-synth you will find quite a few of existent hardware- and software-solutions and additional information of the specifics for each of those “MPE-senders” or “MPE-receivers”.

Many of this may be written in “marketing-speech”, though.
So that effectively different terms are often use to describe the same things, basically.

A common way to describe MPE actions is:

"Strike"	Key on intensity / Attack-Velocity
"Press"	Aftertouch / Channel Pressure
"Slide"	CC 74 (normally vertically)
"Glide"	Pitchbend (normally horizontally)
"Lift"	Key off intensity / Release-Velocity

Sometimes this also will be referred to as the 5 dimensions of polyphonic expression.

Because, as you’ll will see later, our DIY device will focus on sending continuous event-data we will be using “Press”, “Slide” and “Glide” mainly and will use the more common terms Aftertouch, Continuous Controller 74 and Pitchbend.

For further information, you may want to look here:
https://www.midi.org/articles-old/midi-polyphonic-expression-mpe

If you want to download the official documentation here, you have to sign in at the MIDI-association first, though.

The spec itself admittedly is quite technical in some places.
But all this should not be required to understand or build and use the DIY-device as described further below.

Goals of this project

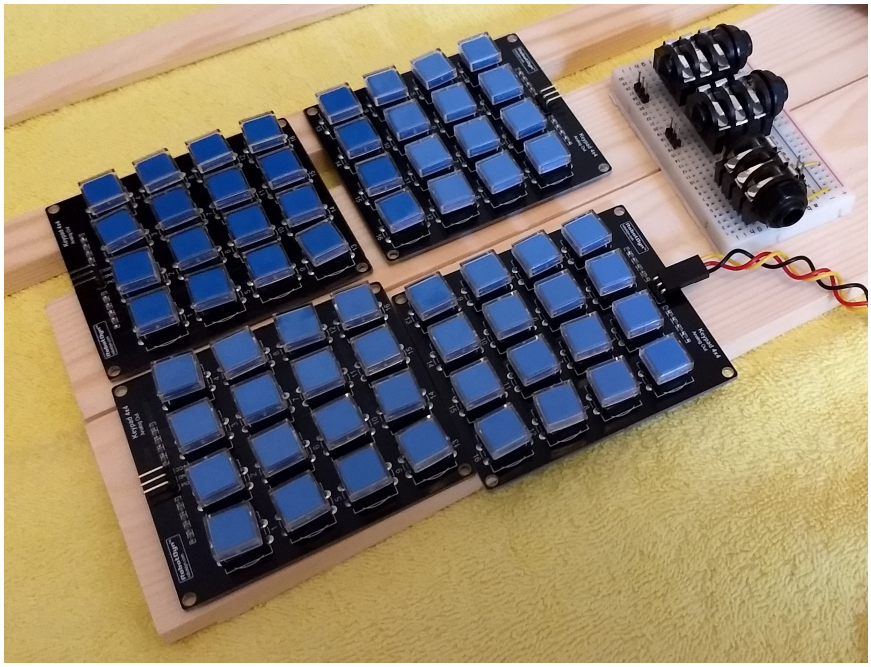
When thinking about what the MPE-controller should be about, several aspects to be considered came to mind – see below, more or less in order of priority:

- Considerably low price, similar to the cost of a Raspberry Pi for instance.
- Small in size, so that it would take up little space on a table and to make it portable.
- Design suitable for every-day operation, not “too ugly”.
- Reliable and sturdy, especially in case it would be used for a live-performance.
- Easy to build, possibly without any soldering or need of other special craftsmanship.
- Required parts widely available, so that they are easy to be sourced.
- Usable for left-handed as well as right-handed operation.
- Software based on well-tested libraries, so it would be easy to program.
- Self-contained device, so that additionally only a computer would be required.

Looking back, all the above mentioned issues can be considered as solved.
Yet compromises had to be made to combine the last aim with the other ones.

I could not find any real small and yet cheap hardware able of triggering notes (some sort of array of buttons for instance) to be attached to a single microcontroller in order to keep the MPE-DIY-controller to be one small device.

Below you can see the button-arrays, that did not make it into the current design along with jacks to attach foot-pedals to. The decision was taken after trying them out, so actually to make the software work for this was not the issue.



Details of the current compromises on this, to still reach all the aforementioned targets, will be described in further detail below.

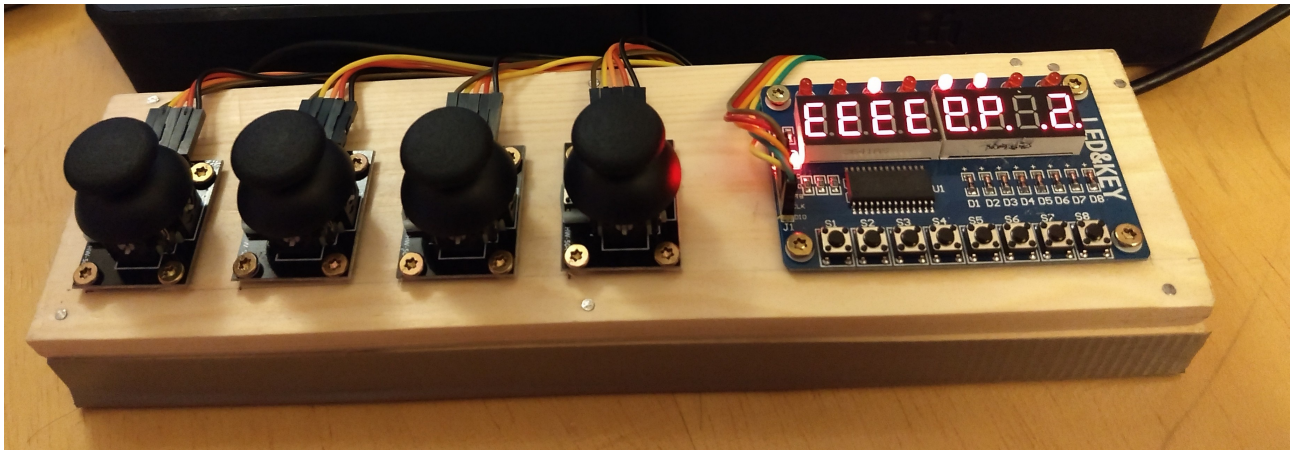
Impementation Ideas/Techniques:

The basic idea was, to provide as much of the MPE-information per channel via cheap joysticks and have a way to assign each joystick to one “string” which would be represented as a separate MIDI-channel to separate the events per note.
To keep things as simple as possible, yet flexible and still manageable 4 joysticks seemed a reasonable number, as then the joysticks could be controlled by the fingers of one hand and still extended (4-voice) chords would be playable.

So because of this and also as they can be bought for really little money as replacement-parts for game-controllers, “thumb-joysticks” where the hardware of choice, here.
To play the notes per “string”, an external matrix-button-controller was chosen, which could provide a “fretboard” via two rows of 4 columns with 8 lines on an 8*8 keypad.
The keypad currently used here does not provide velocity information, but there are other models available that at least have note-on velocity. As this information is transported unchanged, notes with velocity-information would be processed correctly as well.
To make the unit self-contained, the idea was to build in some small button-arrays in case such a MIDI-keypad is not available. But it turned out that as low-cost devices it’s not possible to combine 4 of the 4x4 button-arrays to a 8X8 aray. Alternatively with this solution, each 4x4 keypad would be assigned to one separate midi-channel.
Also optionally, a swell-pedal and footswitches can be connected to modify various things.
To select from various possible settings 8-button based control-panel is attached.
To report the current status of the device, 8 LEDs and a 7 segment-display with 8 digits is used. The different modes of control-data to be assigned to the joysticks will be described below.

In addition to a button-array attached via MIDI, there also is an option to hook up normal keyboard and treat 4 octaves as separate “strings”, i.e. being associated to the 4 joysticks.
In the future there also is the plan for an additional device adding the missing build-in keys.

A more stable version than the first prototype is built into the aforementioned wooden box and using a separate control-panel as in the picture below, current size is 25*7*3 cm.



Of course the design could be much more sophisticated but I kind of like the rough look of the components contrasted by the wood.

Microcontroller / Integration:

As the microcontroller for this project a Teensy LC (“Low Cost”) was selected.
This is an Arduino-compatible processor, but with a different main chip and slightly different and/or extended library support. Still most of the Arduino stuff can be used!
For more details about the Teensy, see here: https://www.pjrc.com/teensy/

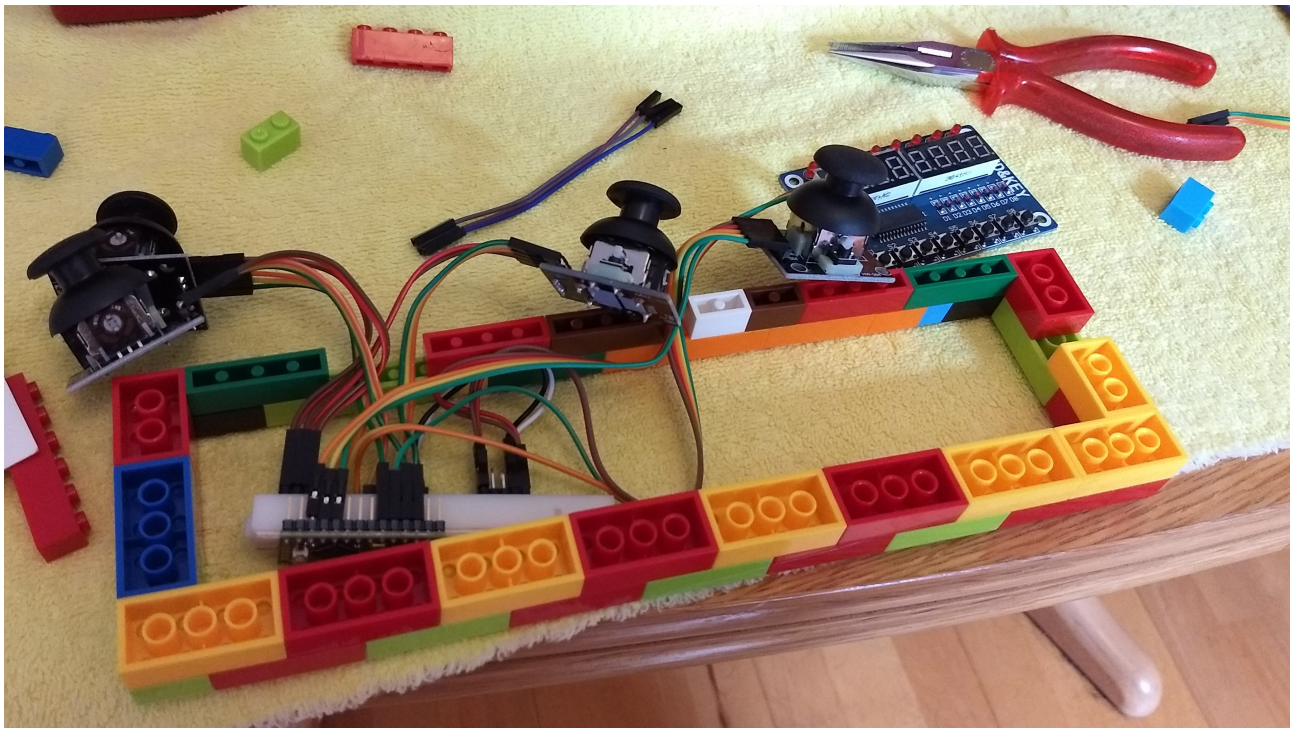
There are many advantages (in addition to the reasonable price) with this hardware for the tasks to be done:

- MIDI over USB, so no separate MIDI-ports or computer-MIDI-interface necessary
- Enough analogue inputs for the given purpose are available
- Small in size to be integrated with the joysticks into a small enclosure
- Fast enough processor for the tasks with sufficient space for variables needed
- Very well documented

Keys can be routed via MIDI-in over USB to the Teensy, MIDI-out would be send via Midi-out from the Teensy via USB.

Below you can see how I intermediately experimented to find an appropriate case using toy bricks. All signal-cables (so called female-to-female Dupont-Wires) are going from the Teensy to the hardware used.

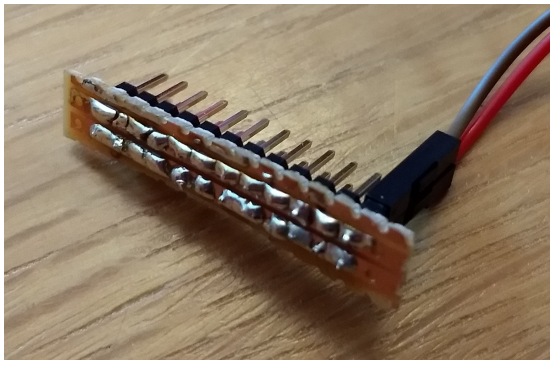
In between the Teensy’s header-pins you may note a strip that I lifted from the breadboard as to be seen in the first picture of this documentation. It simply is used as a power-supply-rail (Ground / 3.3V) that would aliment each of the hardware devices with current.

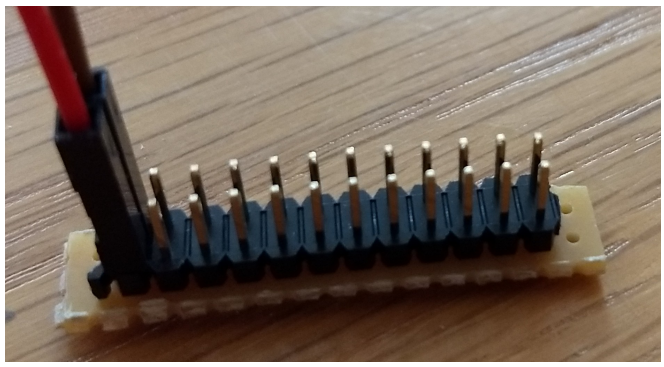


Building instructions (Background information) / Sourcing components:

Basically all connections can be done using so called “Dupont” wires with female to female connectors, max 20 cm long.
Those connectors would go the Teensy’s pins or power-supply on one side and the joysticks or operation-panel on the other.
A common “power-rail” can be used to supply +3.3V and Ground to the hardware on top.

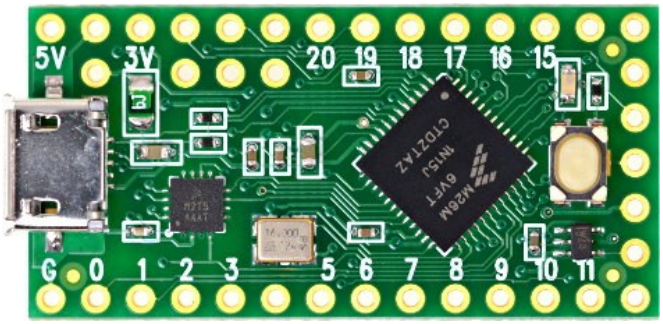
If you want a bit more permanent design as the before-mentioned breadboard-strip, you for instance can solder a power-rail using double header-pins and a piece of a copper-plated experimentation board. The wires you can see in the pictures below then would go to the pins marked 3V and Ground on the Teensy, the still empty connectors later-on would supply the joysticks and the operation-panel.





Caution:
Please be aware, that the Teensy LC (contrary to other Teensys) is not 5V tolerant.

So be sure to source the joysticks with the pin marked **3V**, and **not** to Vin/5V because this current will flow back to Teensy’s analogue input-pins to measure the degree of movement.

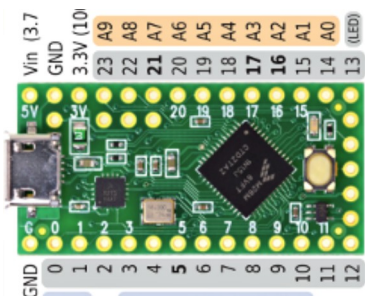


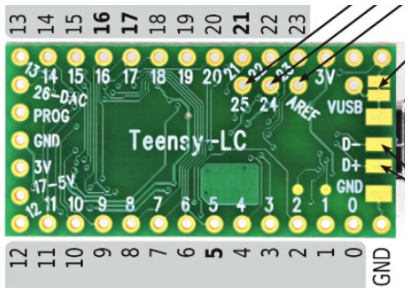
Even when using another Teensy, that is 5V tolerant on the inputs, you should stick to 3.3V as you power-supply, because otherwise the joysticks may need recalibration in the source-code.
More details on the Teensy LC in particular (including the picture above) see here:
https://www.pjrc.com/store/teensylc.html

To buy a Teensy LC, as well as the 4 joysticks and control-panel, please go to the “DIY Electronic Shop” of your choice. You will also need at least 5 sets of multi-coloured Dupont-Cables (female to female connectors), 20 cm in length.
Alternatively you may directly search the net for “***Teensy LC***”, ***“PS2 Joystick Arduino”*** ***“Dupont cables“*** and “***Led&Key***” to order the microcontroller, joysticks, cables and panel.

Building instructions (Connections):

For details on Teensy LC's pinouts, including the top/bottom layouts below please refer to:
https://www.pjrc.com/teensy/pinout.html





→ *External devices on the left-hand side - pins of the Teensy on the right-hand-side:*

Power supply:

Ground	-	GND from Teensy	(power-rail as described above)
+3.3V	-	3.3V from Teensy	(power-rail as described above)

Joystick 1:

VRx	-	Pin 23/A9	(horizontal control)
VRy	-	Pin 22/A8	(vertical control)
SW	-	Pin 01	(switch when joystick is pressed down)
+5V	-	+3.3 V	(no worries, 3.3V works too) applicable to all joysticks!
GND	-	Ground	(we found common ground!) applicable to all joysticks!

Joystick 2:

VRx	-	Pin 21/A7	(horizontal control)
VRy	-	Pin 20/A6	(vertical control)
SW	-	Pin 02	(switch when joystick is pressed down)

Joystick 3:

VRx	-	Pin 19/A5	(horizontal control)
VRy	-	Pin 18/A4	(vertical control)
SW	-	Pin 03	(switch when joystick is pressed down)

Joystick 4:

VRx	-	Pin 17/A3	(horizontal control)
VRy	-	Pin 16/A2	(vertical control)
SW	-	Pin 04	(switch when joystick is pressed down)

Control Panel:

STB	-	Pin 15	(Strobe for TM1638 chip)
CLK	-	Pin 14	(Clock for TM1638 chip)
DIO	-	Pin 13	(Digital I/O for TM1638 chip)
VCC	-	+3.3 V	(GND from Teensy via power-rail)
GND	-	Ground	(3.3V from Teensy via power-rail)

Loading the software (MPE DIY firmware for Teensy) to the microcontroller (Binary):

You can find the binary on Github:
https://github.com/Visuelle-Musik/MPE-DIY-Device

To load it you need a tool called Teensy Loader Application which can be found here:
https://www.pjrc.com/teensy/loader.html

For detailed information on how to install and use it on your platform (Windows, MacOS, Linux, BSD Unix) please also refer to one of the links below:

https://www.pjrc.com/teensy/loader_win10.html

https://www.pjrc.com/teensy/loader_mac.html

https://www.pjrc.com/teensy/loader_linux.html

Simply put the file **MPE-DIY-Device-*.ino.hex** on your computer somewhere and follow the instructions according to the links above appropriate to your platform about how to load it, send it to and activate it on the Teensy.

Building MPE-DIY-firmware for Teensy from source and/or modifying the software:

You can find the sourcecode on Github:
https://github.com/Visuelle-Musik/MPE-DIY-Device

To keep it easy to maintain and debug if necessary, it was done in C++ implementing special MPE-related classes.
There is inline-tracing going out via serial to your computer-screen, that can be turned on if needed. If you want to learn more about the details, please refer to the source-code itself.

To modify the sourcecode, for instance if you want to calibrate/optimize it for your personal joysticks you need the Arduino IDE plus the extensions for Teensy.
The package is commonly known as Teensyduino, for more information see here:
https://www.pjrc.com/teensy/teensyduino.html

For detailed information on how to install it on your platform (Windows, MacOS, Linux, BSD Unix) please also refer to:
https://www.pjrc.com/teensy/td_download.html

The MPE-DIY source-code hopefully should be self-documenting, at least if you are familiar with Arduino / Teensy as well as with C/C++ and have read this documentation on the concepts already. It's free software under the terms of GNU General Public License 3.

For those familiar with C++, here are some explanations about the structure of the code:

There are 3 main classes involved:
class TM1638_MB : public TM1638
class MPE_CTRL_SETTINGS : public MPE_CTRL
class MPE_CTRL

The first class would build an abstraction for the control-panel (leds&keys) of MPE-DIY.
The second class would mainly handle and remember events coming in once, including keyboards, matrix-buttons, pedals and the control-panel.
The third class would mainly handle and remember events coming in from the joysticks.

Some hopefully pragmatic compromises had to be made within the object-model to integrate methods from classless Arduino / Teensy related methods including the main loop and Midi-sending or Midi-receiving functions.
The kind of trickery here was to treat them like static methods of a top-level object and also make several methods of the MPE-classes static, so they for instance still can be used as call-backs/event-handlers for those methods.

Another trick was to make the main objects static, so they can "live on" inside the main loop. Kind of bypassing setup(), they also still would be initialized before the main loop() starts to run. To avoid any global and even module-static objects (i.e. variables), pointers to these objects would be passed on between methods where ever needed.

Building MPE-DIY-firmware for Teensy from source / Additional libraries

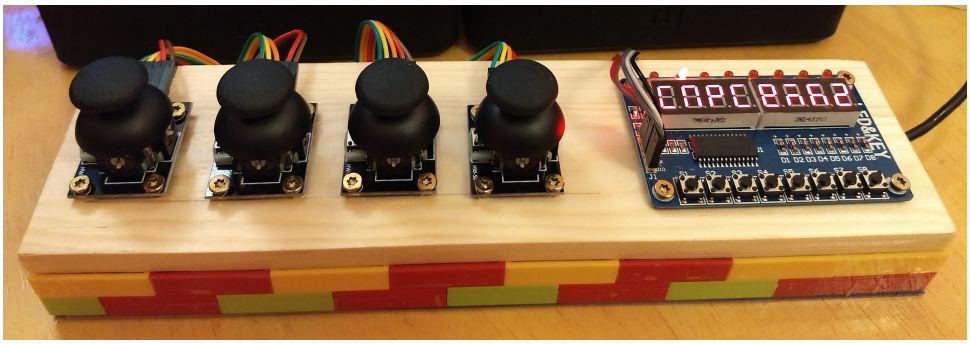
To build from source you also require the TM1638 Library to be found here:

https://github.com/rjbatista/tm1638-library

Concerning on how to integrate the library to your Arduino / Teensy environment please refer to the instructions on that site as well.

Operation:

On the left hand side there are 4 finger-joysticks, each would send expression-data for one of the 4 channels / notes that can be modified.
When a Joystick is pushed down, it's current state will be "frozen" until it is pushed again.



On the right hand side, there is a small panel with 8 buttons, 8 Leds and 8 7-segment displays. The 4 leds on the left side will be lid, whenever the equivalent joystick (left to right) is frozen (see above). The 4 leds on the right will be lid as long as note-data is received, per channel / notes from left to right.
The user-interface is divided into 8 columns, so that each button corresponds to the small display above. The four columns on the left represent the status and possibilities for each joystick, left to right. For each joystick a separate operation-mode can be selected (details see below.)
The four columns on the right have the following functions, by pushing the button below several times, so this would be the options per button:

1. Select Mode for Joystick 1 (Details see below)
2. Select Mode for Joystick 2 (Details see below)
3. Select Mode for Joystick 3 (Details see below)
4. Select Mode for Joystick 4 (Details see below)

5. Select tuning: E (E,A,D,G) – D (D,A,D,F#) – C (C0,C1,C2,C3)
When buttons on the keys-button-array or on the keyboard associated with the MPE-controller are hit, per channel ("string") the lowest key will be like described in the tuning, and go upwards in half-tones, as on a normal guitar or keyboard.

6. Select Mono- or Polyphonic mode: M / P
In Mono-mode only one key will play at a time, high tones have priority, lower keys get triggered again if still hold when higher keys are released – very much as with an analogue synth or a guitar.
Polyphonic mode will act like it is standard with a musical keyboard. Then the modulation from the joystick will be applied at once on all keys of one channel.

7. "Shift-Button", will reverse the direction of selection for any other button

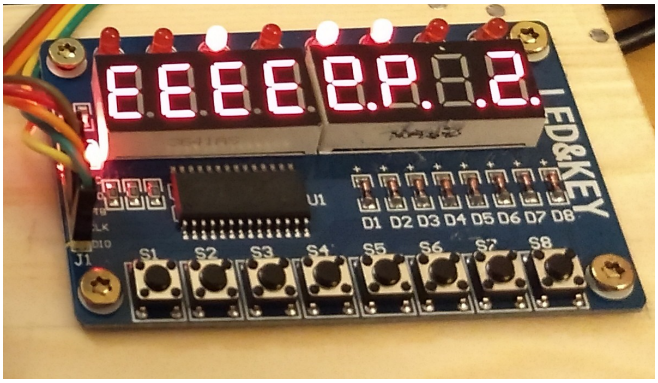
8. Increase Master-pitch by one octave: 0 - 4

Special Operation-Panel Modes

Button No. 7 / “Shift-Button” also works as “octave down” if used alone, but for technical reasons lowering the octave will happen not directly, but when the button is released .

If **Button 1 and 8 pressed at the same time**, there is a hidden function:
“Jitter” will be turned on/off – a ‘J’ will be displayed at Position 7 if on.
If Jitter is off, less data will be send and min and max positions will be stabilised – this is the preferred behaviour! You can try out ‘J’ for some synths that rely on new expression-data with each key pressed and if you want increased / finer joystick action.

If a Sustain-Pedal is used (via the planned future add-on-device and/or via MIDI), the 4 dots on the right 7-segment display will be lid (see below).



If an expression-pedal for CC74 is attached (via the planned future add-on-device and/or via MIDI), the channel it is associated with will be shown by one of the little dots on the left side of the display.

Further details about attaching keyboards and pedals please see below.

Joystick-Operation-Modes / MPE-Modes:

Operation modes can be selected by pressing the left 4 buttons on the operation-panel several times. It's available separately for each joystick and modes are displayed above the buttons (also see picture above).

< Joystick to the left, **>** Joystick to the right, **^** Joystick upwards, **v** Joystick downwards

- *Mode '**E**' - mp**E**-mode:*

< : Pitch down
> : Pitch up
^ : CC 74 0-127
v : Aftertouch 0-127

- *Mode '**N**' - mpe-mode, **N**o Pitchcontrol:*

< : CC 73 0-127 // Useful for synths offering additional MPE ctrl
> : CC 74 0-127 // Can be combined with Aftertouch on this axis
^ : CC 74 0-127
v : Aftertouch 0-127 // Aftertouch, can be combined with CC73 or CC74

- *Mode '**R**' - mpe-mode, "**R**elative controllers" plus pitch:*

< : Pitch down
> : Pitch up
^ : CC 74 64-127
v : Aftertouch 64-127

- *Mode '**L**' - mpe-mode, "**L**arge" controller range:*

< : Aftertouch 64-127
> : Pitch up
^ : CC 74 64-127
v : CC 74 63-0

- *Mode '**F**' - mpe-mode, "**F**ine"-control:*

< : Aftertouch 63-0
> : Aftertouch 64-127
^ : CC 74 64-127
v : CC 74 63-0

Joystick-Operation-Modes / non-MPE-Modes:

As already mentioned: Operation modes can be selected by pressing the left 4 buttons on the operation-panel several times. It's available separately for each joystick and modes are displayed above the buttons (also see picture below).

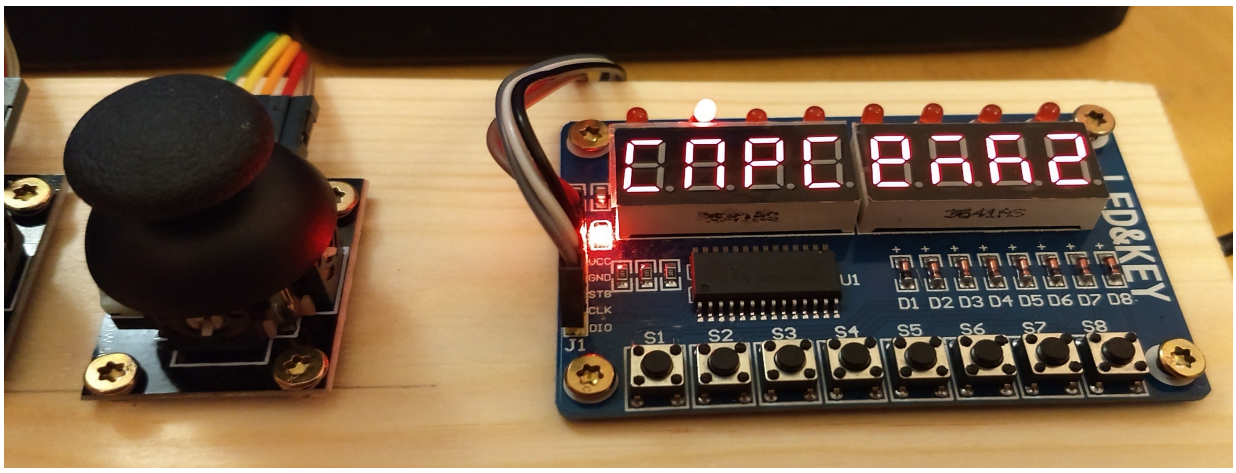
< Joystick to the left, **>** Joystick to the right, **^** Joystick upwards, **v** Joystick downwards

- *Mode '**P**' – non-mpe-mode, incl. **P**itchcontrol:*

< : Pitch down
> : Pitch up
^ : CC 01 0-127 // Modulation wheel, especially for non-MPE synths
v : Aftertouch 0-127

- *Mode '**C**' – non-mpe-mode, **C**ontrollers only*

< : CC 02 0-127 // Breath-Control, especially for non-MPE synths
> : CC 01 0-127 // Modulation Wheel, to be combined with Aftertouch
^ : CC 01 0-127 // Modulation wheel, especially for non-MPE synths
v : Aftertouch 0-127 // Pressure, can be combined with CC01 or CC02



Attaching keyboards, pedals and/or button-arrays to the MPE-controller

Please note: The way incoming events are treated is “channel-selective”, so that note-events from a button-array will be distributed differently to note-events from those of a normal midi-keyboard.
So please be aware, that for automatic recognition the (planned for the future) dedicated button and pedal device has to be connected (via USB or serial, see below).
Since not yet available and/or if more professional equipment is desired, the routing to the MPE-device has to be set up properly in your DAW according to the behaviour as described below.

Global expression information:
On **channel 1** data like Modulation Wheel, Pitchbend, Sustain pedal will be received and passed on unfiltered. This data will normally be regarded as global events for MPE-synths.

Input from keyboard:
On **channels 2-10 i**nput for MPE-controller will be mapped as coming in via a keyboard-device. The first “string”, associated with the first joystick will start at Note C / Number 48, second at C/60, third at C/72 and forth at C/84.

Special functions:
On **channel 11** any note will trigger which channel CC74 should be controlled via the foot-pedal. When pressing the trigger while no note is selected, the function will be disabled. Any controller then will be interpreted as a pedal applying CC74 to the selected channel.

Button Array:
On **channel 16** the notes will be mapped to 4 channels of “guitar-strings” per 2 * 4 columns, “frets” going from the bottom to the top.

Example of keyboard-input on Channel 2/3 plus a button array on Channel 16:

Dedicated hardware to make the DIY-controller “self-contained” (Future project)

As described as one of the goals for this project, the initial idea was to have one self-contained device that would have joysticks, the operation panel and some keys to play the MPE-controller all in one enclosure.

But it turned out if all components would go to one housing the device would become bigger as intended and also the wiring with the microcontroller would become far more complicated, because additional pins would have to be soldered on.
Also the original idea to make the device suitable for left- or right-handed use would be almost impossible to solve. It would be doable with for instance two panels showing the same stuff in different orientations, but this would again make the thing bigger and kind of ugly.
So the compromise here was to have another device in the future to be connected via a separate cable with the MPE-controller.
The cable will simply provide power to the key/pedal device and serial data going back to the MPE-device. Alternatively the device also can be attached via USB, even for stand-alone purposes.

Several pedals can be attached via jack-connectors to the device, information on usage of theses pedals see below.
The output will be in MIDI-format and the data will be identical, regardless if send via direct serial connection or over USB, attaching keyboards, foot-switches or an expression-pedal!

Keyboard:

- 4 button-arrays (Channels 2-5: Notes 36-52)

Footswitches:

- Sustain (Channel 1: CC64 [0: off, 127: on])
- Select-switch (Channel 11: Note 36 [>0 if pressed])

The select-switch will define which channel the expression-pedal will be sending data on. The connection is made depending on the key pressed last from the button-arrays and will be set to the equivalent midi-channel.
Caution: if no key is pressed while the select-switch is pressed, the expression-pedal will be deactivated, that is immediately stop sending any data.

Pedal:

- Expression Pedal (Channels 2-5: CC74 [0-127] or off)

Because this key/pedal device can also be operated stand-alone via USB, and the DIY-MPE device is not directly dependant on it, the idea is to describe it in more detail as a separate project.

MPE-Controller with non-mpe synths / Typical DAW setups

If you have a synth that is not capable of MPE, you still can use it in a pseudo-mpe mode by using the MPE-controller's non-mpe joystick modes and by applying 4 instances of that synth on 4 different channels that the MPE-controller gets routed to in your DAW.

Example-routing in Ableton Live for non-mpe synths:

Other DAWs can do the same in a similar way, please refer to the manual of your DAW.

Because Ableton Live does not directly support MPE, the setup required to still be capable of using it is very similar, the only difference is that only one MPE-capable synth is used as the receiver for the MPE-related signals!