

# Technologie-Evaluierung von Drehgebern und deren Anbindung an eine Java-basierte Potiboard-App

## Table of Contents

Einbindung dieser Arbeit in das laufende Projekt FCC Digital Potiboard .....	2
Rückblick und Motivation .....	2
Skizze des neuen Potiboard-Encoder-Device .....	3
Vom alten Potiboard-Prototyp übernommene Technologieentscheidungen und neue Wege .....	4
Generelle Hardware-Komponenten-Architekturüberlegungen .....	4
Komponentengruppen .....	4
Inkrementzähler-Übertragung über Netzwerk oder USB .....	5
Netzwerkbasierte System-Architekturen und Technologien .....	6
Testsystem mit <b>Spring Webflux</b> .....	6
Testsysteme mit <b>ZeroMQ</b> und <b>MQTT</b> .....	7
USB-basierte System-Architekturen und Technologien .....	8
Testsystem mit USB-Serial .....	8
Testsystem mit USB-HID .....	9
Testsystem mit der USB-MIDI .....	9
Grundlegende zentrale Anforderungen (Stichwortliste) .....	10
Tabelle Technologiebewertung .....	11
Fazit .....	11
Weitere bearbeitete aber nicht ausgeführte Themen (stichpunktartig) .....	13

# Einbindung dieser Arbeit in das laufende Projekt FCC Digital Potiboard

Die hier beschriebene Technologie-Evaluierung dient als Grundlage zur Entscheidungsfindung zu den in der [Abbildung: Status FCC Digital Potiboard](#) markierten *Major Milestones* und *Subprojects / Tasks*.



<b>Project</b>	FCC Digital Potiboard																					
<b>Project Lead</b>	S. Reimann																					
<b>Status Date</b>	08.07.2022																					
<b>Project Description</b>																						
	<p>Development of a digital potentiometer board control for FCC - Focus: UNILAC operation - But also generally as an option for other linear accelerators or any parameters, where a rotating controller makes sense. The project includes the incremental encoder hardware and the software application.</p>																					
<b>Project Goals</b>																						
	<ol style="list-style-type: none"> <li>1. Replace the outdated UNILAC Potiboard control with a modern version</li> <li>2. compatible with fully digital control room (FCC)</li> <li>3. Full control system integration</li> <li>4. Be ready on time for the move to FCC</li> <li>5. Production of relevant spare parts</li> </ol>																					
<b>Major Milestones</b>																						
Q3/2022	Specification revision and approval																					
Q4/2022	technology decision - incremental encoder hardware and communication protocol																					
Q4/2022	decision regarding use of control system stack (FESA or LSA or else) to be able to ensure the required performance																					
Q3/2023	Hardware prototype ready																					
Q1/2024	Software prototype ready																					
Q3/2024	Live test in HKR (e.g. for HEST magnets)																					
<b>Subprojects / Tasks</b>																						
	<ul style="list-style-type: none"> <li>• Specify the system ✓</li> <li>• Specification approval -&gt; <b>open</b></li> <li>• technology decision for incremental encoder -&gt; <b>in progress</b> ↗</li> <li>• technology decision software stack -&gt; <b>in progress</b> (FESA, LSA or else)</li> <li>• general decision on UI -&gt; <b>open</b> (existing or new app + developing group) ↗</li> <li>• build hardware prototype -&gt; <b>open</b></li> <li>• build software prototype -&gt; <b>open</b></li> <li>• FAT → serial production -&gt; <b>open</b></li> </ul>																					
<b>Ressource Profile</b>																						
	<table border="1"> <thead> <tr> <th>Year</th><th>Estimated costs [k€]</th><th>Personnel [person months]</th></tr> </thead> <tbody> <tr> <td>2022</td><td>1</td><td>2</td></tr> <tr> <td>2023</td><td>4</td><td>12</td></tr> <tr> <td>2024</td><td>15 (initial equipment)</td><td>3</td></tr> <tr> <td>2025</td><td>10 (spare parts)</td><td>3</td></tr> <tr> <td>2026</td><td>0</td><td>0</td></tr> <tr> <td><b>Sum</b></td><td><b>30.000 Euro</b></td><td><b>20 person months</b></td></tr> </tbody> </table>	Year	Estimated costs [k€]	Personnel [person months]	2022	1	2	2023	4	12	2024	15 (initial equipment)	3	2025	10 (spare parts)	3	2026	0	0	<b>Sum</b>	<b>30.000 Euro</b>	<b>20 person months</b>
Year	Estimated costs [k€]	Personnel [person months]																				
2022	1	2																				
2023	4	12																				
2024	15 (initial equipment)	3																				
2025	10 (spare parts)	3																				
2026	0	0																				
<b>Sum</b>	<b>30.000 Euro</b>	<b>20 person months</b>																				
<b>Risks, Boundary Conditions and Comments</b>																						
	<p><b>Major Risks:</b></p> <ul style="list-style-type: none"> <li>- turnaround time too slow for adequate UNILAC control</li> </ul> <p><b>Concerned departments:</b></p> <ul style="list-style-type: none"> <li>- ACO (FE, AP, IN)</li> <li>- OPE (APS)</li> </ul>																					
	<p><b>Boundary Condition:</b></p> <ul style="list-style-type: none"> <li>- must be functional before move from HKR to FCC</li> <li>- existing potiboard must kept functional until move to FCC is complete</li> </ul> <p><b>Comments:</b></p> <ul style="list-style-type: none"> <li>- to be clarified: Responsible group for hardware maintenance</li> </ul>																					

Figure 1. Status FCC Digital Potiboard

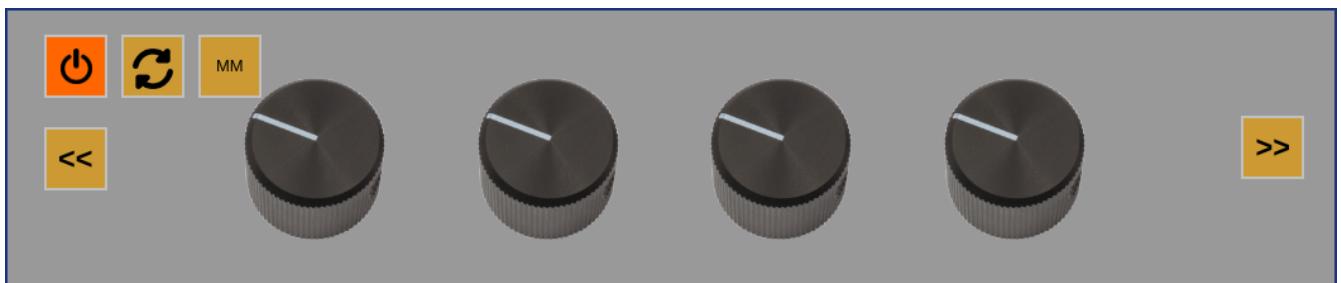
## Rückblick und Motivation

Der vor ca. 6 Jahren entwickelte Potiboard-Prototyp wurde in einem vorhergehenden Potiboard-Meeting als unzureichend in seiner Technologieauswahl eingestuft. Insbesondere der Einsatz der "closed source library" des damals benutzten Phidget-Mikrocontroller, der zur Verarbeitung der Encodersignale dient, wurde als kritisch eingestuft. Die benötigte Bibliothek des Phidget läuft zudem nicht im "Userspace" auf Linux, was aus system-administrativer Sicht ein Nachteil ist.

Die damaligen Anforderungen wurden beim [alten Potiboard-Prototyp](#) über eine hohe Integration der beteiligten Komponenten gelöst. Drehgeber, wie Potiboard-App, gesteuert über einen Touchscreen, befanden sich in einem Gehäuse mit Netzwerkanschluss, über den der Gerätezugriff direkt über [JAPC](#) erfolgte.



## Skizze des neuen Potiboard-Encoder-Device



Funktionsüberblick aus der Spezifikation

- Button functions are:
  - scroll back and forth (per button press by 4 devices to the right or left.  
In case of individual assignment, move one device to the right or left)
  - deactivate the encoders (deactivation deselects all devices and triggers the LSA data supply)
  - change the parameter view ⇒ volts, BRHO et cetera
  - activate and deactivate the master mode (first partner knob controls both devices, second (third, fourth) partner is ignored)
  - change the increment

— Spezifikation: F-FO-CMD-en-0009\_DS\_Potiboard\_v4\_inprogress.docx

# Vom alten Potiboard-Prototyp übernommende Technologieentscheidungen und neue Wege

- Optische "Rotary Quadrature Encoder" wurden wieder, wie beim alten Prototypen, wegen ihrer Signalqualität, Zuverlässigkeit und Verfügbarkeit eingesetzt. Auf kugelgelagerte Modelle wurde diesmal verzichtet (Haptikgründe wegen zu hoher Leichtgängigkeit). Andere aktuelle Auswahlkriterien sind Modelle mit 32-128 Pulsen pro 360-Rotation, keine Zahnrung und auf 5 V ausgelegte Signalverarbeitung. 3.3V Modelle, die alle anderen Kriterien auch erfüllen, waren in den letzten Wochen nicht lieferbar.
  - Getestete-Encoder sind:
    - Grayhill 63R128, Stückpreis: 100 \$
    - Bourns ENA1J-B28-L00128L, Stückpreis: 60 \$, gleiches Modell wie beim alten Prototypen.

Durch den Einsatz von modernen Mikrocontrollern, deren Spannung an ihren I/O Kanälen häufig auf 3,3 V limitiert ist (anstatt 5V), schränkt sich die Auswahl der möglichen Endcoder-Modelle deutlich ein. Eventuell müssten die Encoder-Ausgangsspannungen an den Eingängen der Mikrocontroller mit Pegelumsetzern (Level-Shifter) angepasst werden, wenn aus Gründen von Lieferschwierigkeiten 5 V Encoder-Modelle eingesetzt werden müssen.

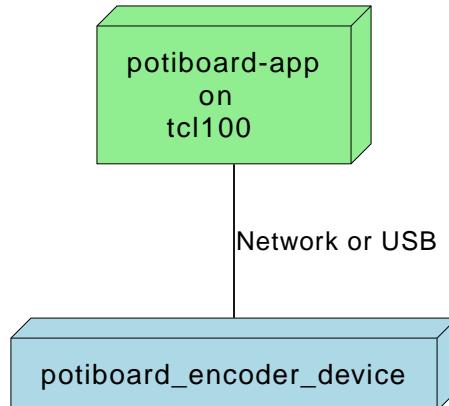
- Es wurden neue Mikrocontroller evaluiert, die die Inkremente der bis zu acht (8!) Encoder, ohne spürbare Zeitverzögerung, weiterverarbeiten können sollen. Statt des im alten Prototypen verwendeten Phidget-Mikrocontroller (1047\_1), wurden folgende Mikrocontroller stattdessen in die Auswahl genommen:
  - Raspberry Pi 4, Stückpreis: 70 \$
  - STM32H7, STM32F7, Stückpreis: 70 \$
  - Teensy 4.1 (Arduino kompatibel), Stückpreis: 40 \$
  - Raspberry Pi Pico, Stückpreis: 8 \$

Alle diese Mikrocontroller sind verbreitete Modellarten (Ersatzteilversorgung scheint vorerst gesichert) und lassen sich mit Open-Source Software betreiben. Der Mikrocontroller-Code zur Weiterverarbeitung der Encoder-Inkremente muss in C, Python oder Assembler geschrieben und gewartet werden. Das gleiche gilt für die verschiedenen Übertragungstechnologien zur der Java-basierten Potiboard-Applikation, wenn keine zusätzliche Rechnerschicht eingeführt wird.

## Generelle Hardware-Komponenten-Architekturüberlegungen

### Komponentengruppen

## General topology for connecting a potiboard encoder device



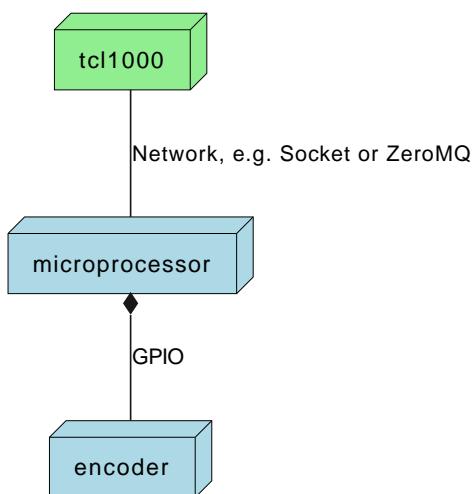
General Deployment Diagram, A. Bloch-Späth, M. Stein

# Inkrementzähler-Übertragung über Netzwerk oder USB

GSI/FAIR 03.11.2022

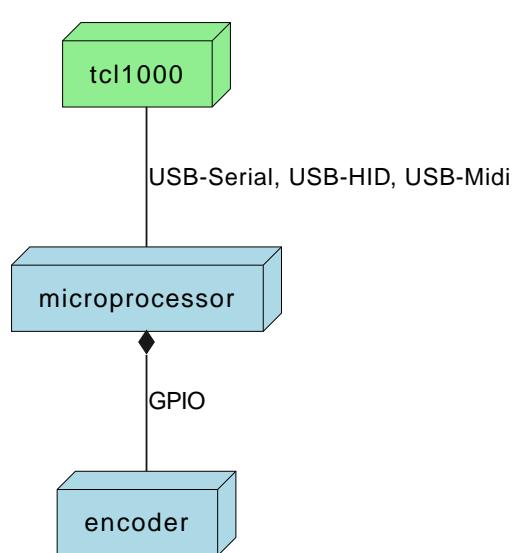
GSI/FAIR 03.11.2022

**Network based encoder counter transmission**



Deployment Diagram Potiboard Network, A. Bloch-Späth, M. Stein

**USB based encoder counter transmission**



Deployment Diagram Potiboard USB, A. Bloch-Späth, M. Stein



### USB Nachteil

USB ist auf eine **maximale Kabellänge** von 5m spezifiziert. Mit guten Kabel und/oder Repeatern sind vielleicht bis zu 10m möglich.



### USB Vorteil

USB ist schneller als die Netzwerkuübertragung, insbesondere bei der Übertragung von kleinen Datenpaketen. Die USB-Protokolle haben keinen Adressierungs-Overhead. Vieles ist dadurch einfacher, z.B. müssen Sender (Potiboard) und Empfängeradresse (Potiboard-App) nicht konfigurierbar programmiert werden.

### *USB Vorteil*



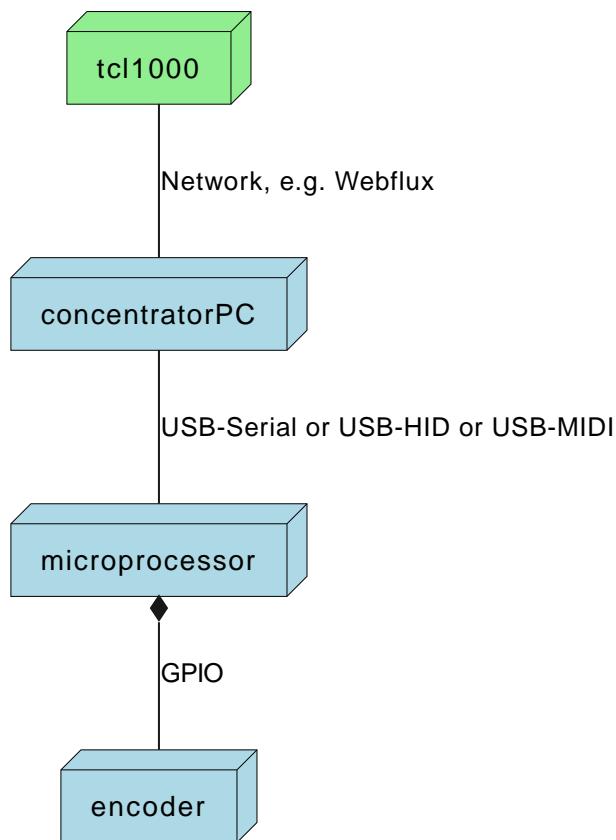
USB liefert out of the box ausreichend Strom für Drehgeber und Mikroprozessor. Beim Netzwerk müßte zusätzliche Hardware (z.B. PoE) hinzugefügt werden, wenn ein Stromnetz-Anschluss vermieden werden soll (USB nur zum Stromanschluss ginge natürlich auch).

# Netzwerkbasierte System-Architekturen und Technologien

## Testsystem mit **Spring Webflux**

GSI/FAIR 08.07.2022

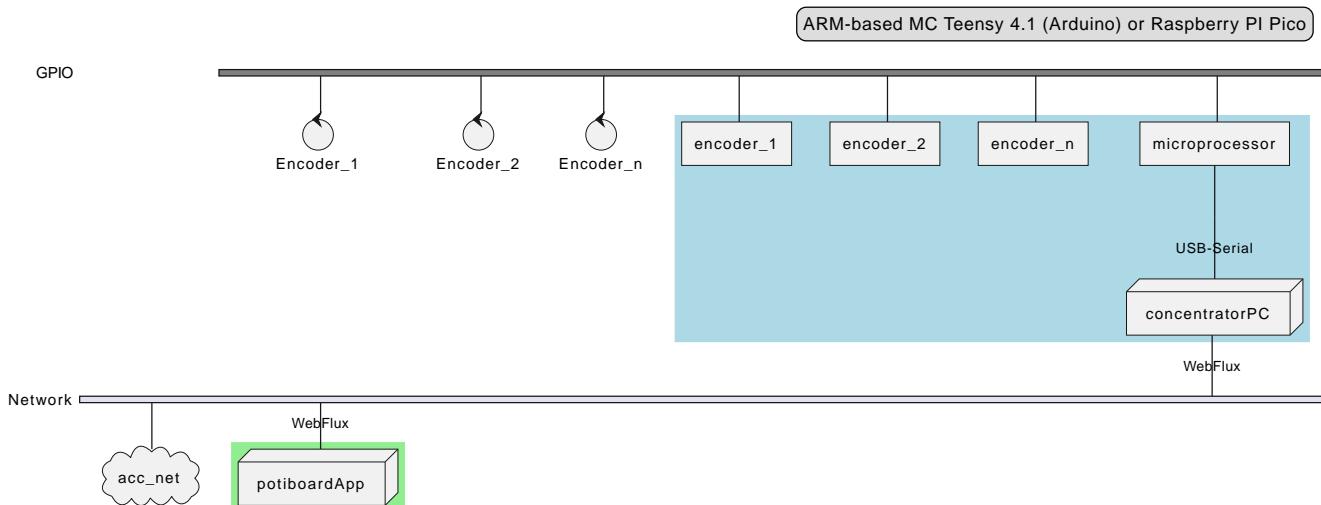
### Reference Implementation with Spring WebFlux



Deployment Diagram Potiboard Webflux, A. Bloch-Späth, M. Stein

Figure 2. UML-Komponenten Diagramm Netzwerkübertragung with **Spring Webflux**

### Reference Implementation with USB-Serial and Spring WebFlux



System/Network Diagram Potiboard 1., A. Bloch-Späth, M. Stein

*Figure 3. Test-Implementation 1*

Es wurde ein Referenzsystem, wie im oberen Bild dargestellt, auf Basis eines Teensy 4.1 Mikrocontrollers entwickelt, der die Inkremente der Encoder in hoher Geschwindigkeit bis in eine Beispiel-JavaFX-Applikation weiterreicht.

Die im Referenzsystem eingesetzte Datenübermittlungstechnologie basiert auf der Technologie **Spring Webflux** und dem "Reactive Toolkit" **Project Reactor**. Sie wurde ausgewählt, da sie der "GSI Controls Applicationsservice-Technologieauswahl" entspricht, die für die Operating-Applikationen im FCC und HKR eingesetzt werden soll und teilweise schon eingesetzt wird.

Ein Nachteil und in mancherlei Hinsicht sicher auch Vorteil dieser Architektur ist die Einführung eines java-basierten **Webflux**-Servers (siehe Bild **EncoderPositionsServerPC**), der ein PC-System mit Controls-konformen OS sein sollte. Es ist also eine Schicht (*Tier*) notwendig, um die Inkremente der verschiedenen Encoder im **WebFlux**-Format zu versenden.

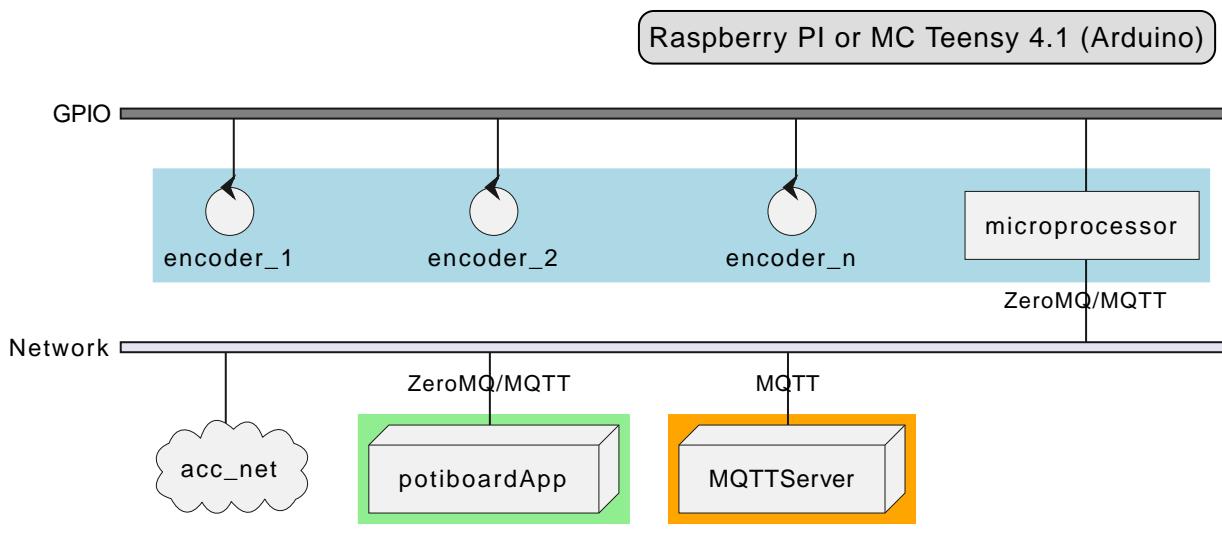
Auf der Habenseite dieser Architektur steht die Anpassbarkeit und Wartbarkeit nach den Richtlinien der Controls-Abteilung und damit eine sichere, kontrollierbare Netzwerkkommunikation im ACC-Netzwerk und keine Insellösung im ACC-Netz.

Die weiteren im Anschluss besprochenen netzwerkbasierten Architekturen könnten den Einsatz eines weiteren Rechners, wie der des Konzentrator-PCs überflüssig machen. Die Instandhaltung des Rechners, so wie die Wartung des Betriebssystems (z.B. Rocky Linux), erzeugt wiederkehrende Kosten.

## Testsysteme mit **ZeroMQ** und **MQTT**

*Test-Implementation 2*

## Implementation with ZeroMQ or other messaging libraries



System/Network Diagram Potiboard 2., A. Bloch-Späth, M. Stein

Ein Kandidat für eine einfachere Architektur ist zum Beispiel die Technologie **ZeroMQ**, die sich mit einem Raspberry Pi 4, wie getestet, leicht umsetzen lässt.

Sehr interessant ist auch die **MQTT**-Technologie, die allerdings die Notwendigkeit des Aufsetzens eines **MQTT**-Servers nach sich ziehen würde und somit den Vorteil der Kostenersparnis zumindestens teilweise wieder verliert.

# USB-basierte System-Architekturen und Technologien

## Testsystem mit USB-Serial

### *Mikroprozessor-Seite*

Die USB-Serial Übertragung wurde hauptsächlich vom Mikroprozessor Teensy 4.1 aus getestet, da dieser auch für den netzwerk-basierten Test mit **Spring Webflux** zum Einsatz kam. Es wurde die Standard Arduino Bibliothek **Arduino.h** für das Schreiben auf der USB-seriellen Schnittstelle eingesetzt.

### *Java-Applikationsseite*

Es wurde die gut gepflegte und verbreitete Java-Bibliothek **jSerialComm** genutzt um die seriellen Datenpakete in der Potiboard-Applikation zu empfangen.

### *Administration, Konfiguration*

Administrationsaufwand ergibt sich für die Java-Applikationsseite auf Linux-Systemen, z.B. auf den **tcl1000**-Maschinen, durch die Notwendigkeit einer Rechtevergabe für den **user**, um unter Linux Zugriffrechte auf die Serielle Schnittstelle zu bekommen.

```
sudo usermod -a -G uucp username
sudo usermod -a -G dialout username
sudo usermod -a -G lock username
sudo usermod -a -G tty username
```

Zusätzlich wird die Installation einer C-Bibliothek auf dem Host-Rechner (z.B. auf den tcl1000-Maschinen) auf der Java-Applikationsseite benötigt.

## Testsystem mit USB-HID

### *Mikroprozessor-Seite*

Wieder kam der Teensy 4.1 zum Einsatz, diesmal mit einer Teensy-spezialen Bibliothek [USB Host Library for Teensy 3.6 and 4.0](#). Mit dieser können USB-HID konform "Rohdaten" bis zu einer Länge von 64 Byte (pro Millisekunde) in einem Datenpaket übertragen und empfangen werden.

### *Java-Applikationsseite*

Auf Java-Seite setzte sich eine sehr leichgewichtige Bibliothek durch, die [Pure Java HIDApi](#). Sie bekam den Vorzug zur [HID4Java](#), die zusätzlich eine C-Bibliothek (z.B. auf den tcl1000-Maschinen) als Abhängigkeit benötigt.

### *Administration, Konfiguration*

Administrationsaufwand ergibt sich für die Java-Applikationsseite auf Linux-Systemen (z.B. auf den tcl1000-Maschinen), die Notwendigkeit einer [UDEV](#)-Regel, z.B. in einer Datei [66-hid-rules](#) im Verzeichnis [/etc/udev/rules.d](#):

```
KERNEL=="hidraw*", ATTRS{idVendor}=="16c0", ATTRS{idProduct}=="042", MODE=="0666"
SUBSYSTEMS=="usb", ATTRS{idVendor}=="16c0", ATTRS{idProduct}=="042", MODE=="0666"
```

## Testsystem mit der USB-MIDI

### *Mikroprozessor-Seite*

Auch hier kam der Mikroprozessor Teensy 4.1 in einem erfolgreichen Test zum Einsatz und wieder reichte, wie beim USB-Seriellen Weg, die Standard Arduino Bibliothek [Arduino.h](#), diesmal für das Schreiben auf der USB-MIDI-Schnittstelle.

### *Java-Applikationsseite*

Die Potiboard-Applikation kann die MIDI-Daten in ihrem Java-Code ohne zusätzliche Abhängigkeiten empfangen, MIDI wird direkt von der [JRE](#) unterstützt und ist enthalten in der [Java Sound API](#).

### *Administration, Konfiguration*

Keine Notwendigkeiten unter Linux.

### *Bemerkung*

Um mit MIDI Datentypen größer als zwei Byte in einem Datenpaket zu verschicken, müssen

sogenannte **Sysex**, oder **MIDI system exclusive messages** benutzt werden. Sie dienen normalerweise zur Konfiguration zwischen den MIDI-Geräten. Bei dieser Art von Nachrichten sind mehr als drei Byte Datenpaketlänge erlaubt. Allerdings können in jedem übertragenen Byte nur 7 Bit genutzt werden (1 Bit ist ein Status-Bit), so dass Konvertierungen vom 8-Bit-System in 7-Bit-System und zurück sowohl auf Mikroprozessor-Seite (C), als auch auf Java-Applikationsseite notwendig sind.

## Grundlegende zentrale Anforderungen (Stichwortliste)

### Komplexität, Lebensdauer und Wartbarkeit der **Hardware**

Die Funktion der eingesetzten Drehgeber und Mikrocontroller muss durch Verfügbarkeit am Markt oder durch Reserveteile-Einlagerung für möglichst mehrere Jahrzehnte mit finanziell überschaubarem Aufwand absicherbar sein. Komplexe Systeme oder eine hohe Anzahl von verschiedenen benötigten Hardwarekomponenten sollte wenn möglich vermieden werden.

### Komplexität, Lebensdauer und Wartbarkeit der **Software**

Die eingesetzte Software auf Mikrocontroller und auf Potiboard-Applikationsseite sollte aus möglichst gut gepflegten und verständlichen Open-Source Projekten mit hoher Verbreitung stammen. Dies kann auch Auswirkungen auf die Wahl des Mikrocontrollers haben. Der notwendige selbst geschriebene Source-Code sollte möglichst einfach wartbar sein. Auf dem Mikrocontroller kommen die Programmiersprachen Assembler, C und Python in Frage, auf der Potiboard-Applikationsseite werden Java-basierte Lösungen preferiert.

### Administrations-, Konfigurationsaufwand

Der Aufwand für zusätzliche Hardware und Software, wie z.B. der KonzentratorPC für **Webflux** oder ein **MQTT-Server** (Linux-Administration, Hardwarepflege) oder zusätzliche Stromversorgungswege als auch der Aufwand für Konfigurationen (Netzwerk-Adressen-Pflege) sollte minimal gehalten werden. Unter diesen Punkt fallen auch notwendige Linux-Anpassungen z.B. auf den tcl1000 Rechnern für den HKR.

### Geschwindigkeit der Signalübertragung der Inkremente der Encoder

Die Geschwindigkeit der Inkremente vom Nutzer über den Drehgeber zum Mikrocontroller und dann in das Java-Programm sollte zwischen max. bei 10 ms (100 Hz) liegen, besser deutlich niedriger.

### Duplex-Signalübertragung, nicht nur für die Inkremente der Encoder in eine Richtung, sondern auch in der Gegenrichtung von der Portiboard-App zurück zum Potiboard-

Encoder-Device.

Um die Benutzererfahrung am Potiboard-Encoder-Device zu verbessern, sollte es technisch möglich sein, Informationen wie Status der Verbindung, oder auch Magnet-Nomenklaturen an das Potiboard-Encoder-Device zu übertragen.

## Tabelle Technologiebewertung

Table 1. Versuch der Einordnung der Stärken und Schwächen der verschiedenen Technologien

Eigenschaft — Technologie	USB	Netzwerk	Hardware	Software	Admin	Geschwindigkeit	Duplex	&sum; *
<b>Webflux</b>		X	*	**	*	**	***	9
<b>MQTT</b>		X	*	*	*	**	***	8
<b>ZeroMQ</b>		X	**	**	**	**	***	11
<b>Socket</b>		X	**	**	**	***	**	11
<b>USB-Serial</b>	X		***	***	*	***	**	12
<b>USB-HID</b>	X		***	**	**	**	**	11
<b>USB-MIDI</b>	X		***	***	***	*	*	11
<b>RS232/RS485</b>					-			
<b>MIDI (DIN)</b>					-			
X = gehört zu, - = ungenügend, * = ausreichend , ** = gut, *** = sehr gut								

Die Tabelle dient nur als Diskussionsgrundlage für die verschiedenen Technologien. Für einen Vergleich wären die verschiedenen Eigenschaften (Spalten) zu gewichten. Die &sum; \* Spalte dient nicht zur objektiven Bewertung.

## Fazit

Wenn USB als Datenübertragungssystem für Potiboard-Prototypentwicklungen vorerst als ausreichend bewertet wird, wäre der technische Vorschlag, für den ersten Prototypen die Encoder-Signale mit einem Arduino kompatiblen Mikrocontroller der Art Teensy 4.1 zu verarbeiten und von diesem aus die Inkrementzählerwerte über das USB-MIDI-Protokoll an die java-basierte Potiboard-Applikation weiterzuleiten.

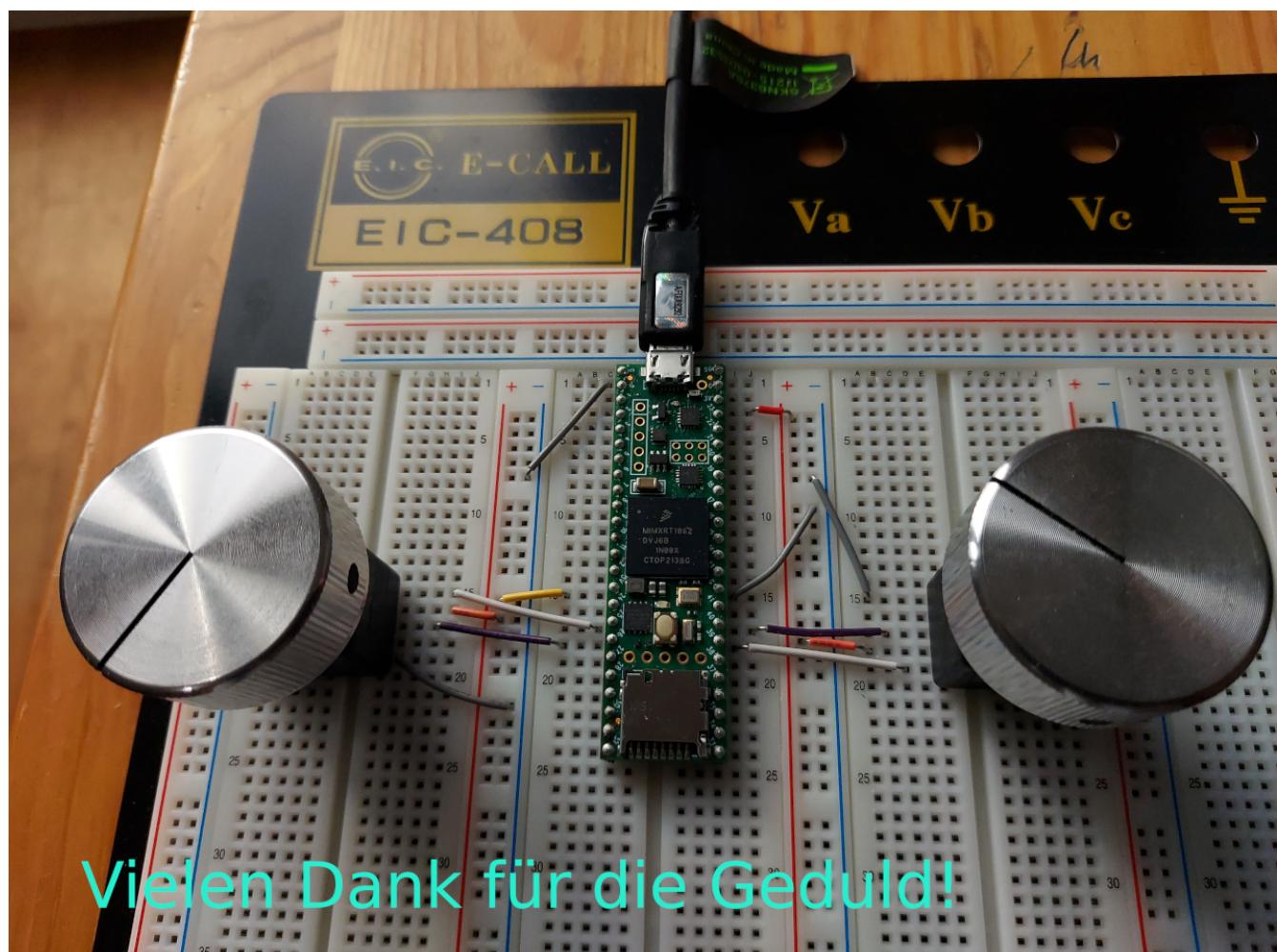


Der Teensy 4.1 ist ein kosteneffizienter, gut verfügbarer und hoch performanter 600 MHz ARM Cortex M7 Mikrocontroller. Seine über die Arduino-IDE leicht einbindbaren Open-Source Bibliotheken sind verbreitet und gut unterstützt. Die in

den Tests eingesetzten Bibliotheken für Encoder sowie die USB-Serial-, USB-HID- und USB-Midi Bibliotheken funktionierten schnell und problemlos.

Das USB-Midi Protokoll bietet als einzige USB-Datenübertragungstechnologie echtes Plug-and-Play an einem Linux-basierten Host (wie z.B. tcl1000). Auf der Java-Seite, also bei der Entwicklung und Wartung der Potiboard-Applikation, wird MIDI direkt von der JRE unterstützt durch die [Java Sound API](#). Es werden auf absehbare Zeit keine zusätzlichen Bibliotheken oder Abhängigkeiten einzubinden sein. Diesen Vorteilen stehen gegenüber eine leicht erhöhte Komplexität bei der Programmierung der Übertragungsdatenpakete und eine niedrigere aber noch ausreichende Datenübertragungsrate.

Wenn USB als Datenübertragungssystem als möglicherweise nicht ausreichend bewertet wird, müsste die Evaluierung der netzwerk-basierten Technologien weitergeführt werden. Eine rein *socket-basierte* Verbindung von einem netzwerk-fähigen Mikrocontroller zur java-basierten Potiboard-Applikation wäre ein begehbarer Weg. Möglich wäre sicher auch die auf das *ZeroMQ-Messaging* basierende Übertragung zwischen Mikrokontroller und der Potiboard-Applikation..



# Weitere bearbeitete aber nicht ausgeführte Themen (stichpunktartig)

- GPS-System zur Uhrensynchronisation für Zeitmessungen mit  $\sim 30 \mu\text{s}$  Genauigkeit.
- 7-Bit pro Byte arithmetische Kodierung zur Darstellung von Datentypengrößen größer als 7 Bit ( $>127$ ) (MIDI-spezial).
- Nutzung der **State Machine** für die GPIO-Kommunikation zur Vermeidung von CPU-Interrupts beim Raspberry PI Nano.