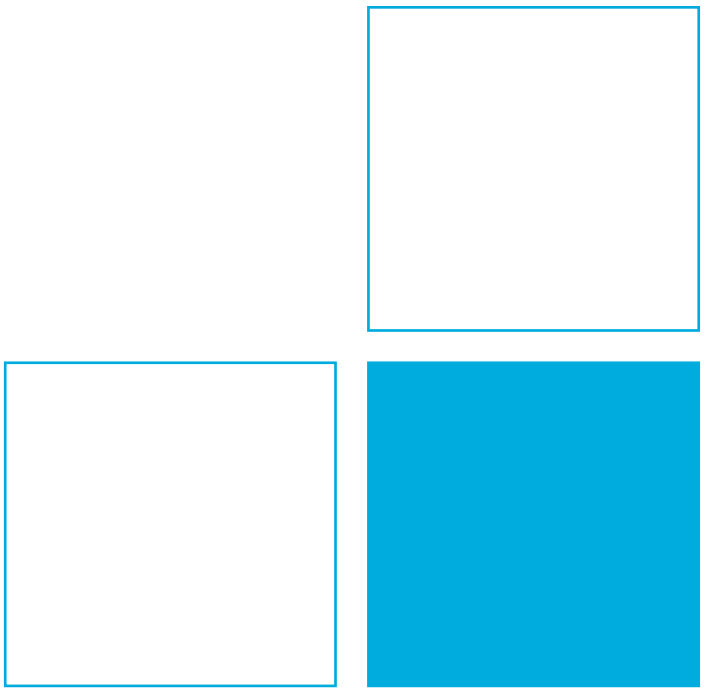
Physikalisch-Technische Bundesanstalt
Nationales Metrologieinstitut

Laborbericht

Gerd Buchholz

Puls-Elektrometer-Verstärker zur parallelen Messung von Strom und Ladung



Physikalisch-Technische Bundesanstalt

PTB-Laborbericht

Gerd Buchholz

Puls-Elektrometer-Verstärker zur parallelen Messung von Strom und Ladung

Braunschweig, Juni 2021

DOI: 10.7795/130.20210610

Empfohlene Zitierweise:
Bucholz, Gerd: Puls-Elektrometer-Verstärker zur parallelen Messung von Strom und Ladung [online]. Braunschweig, Physikalisch-Technische Bundesanstalt, 2021. PTB-Laborbericht. Verfügbar unter: https://doi.org/10.7795/130.20210610

Herausgeber:
Physikalisch-Technische Bundesanstalt
ISNI: 0000 0001 2186 1887

Fachbereich 6.3
Strahlenschutzdosimetrie

Bundesallee 100
38116 Braunschweig

Telefon: (05 31) 592-63 01
Telefax: (05 31) 592-60 15
www.ptb.de

Inhalt

1 EINFÜHRUNG....1

2 TECHNISCHE REALISIERUNG....3

2.1 Generelle Anforderungen....3

2.2 Aufbau....3

2.3 Puls-Elektrometer-Verstärker....4

2.3.1 Vorverstärker....4

2.3.2 Strom-Verstärker....5

2.3.3 Ladungs-Verstärker....5

2.4 Steuerung....6

2.4.1 Spannungs-Messung....7

2.5 Versorgung....7

2.5.1 Akkubetrieb....8

2.5.2 Kammerspannung....8

3 BEDIENUNG....9

3.1.1 Ausgabe über RS232....10

4 KALIBRIERFAKTOREN UND WICHTIGE KONSTANTEN....11

5 TEST-MESSUNGEN....12

6 AUSBLICK....15

7 DANKSAGUNG....16

8 ANHANG....17

8.1 Schaltpläne und Platinen-Layouts....17

8.1.1 Puls-Elektrometer-Verstärker....17

8.1.2 Steuerungsplatine....19

Platinenlayout....20

8.1.3 Versorgung....21

Platinenlayout....22

https://doi.org/10.7795/130.20210610

8.2 Arduino Quelltexte.....23
8.2.1 Arduino Haupt-Programm.....23
8.2.2 Panel.h.....35
8.2.3 Panel.cpp.....36
8.2.4 Button.h.....38
8.2.5 Button.cpp.....38
8.2.6 MillisTimer.h.....39
8.2.7 MillisTimer.cpp.....39
8.2.8 StatFunct.h.....40
8.2.9 StatFunct.cpp.....40

https://doi.org/10.7795/130.20210610

1 Einführung

Eine messtechnische Aufgabe in der Dosimetrie ionisierender Strahlung ist die Messung von Strömen aus Ionisationskammern. Diese Ströme liegen im Bereich von weniger als einem Femtoampere (10^{-15} A) bis zu einigen Nanoampere (10^{-9} A).

Sollen Ionisationskammern im Strahlenschutz eingesetzt werden, ist es vor allem angebracht die aktuell vorhandene Ortsdosisleistung anzuzeigen. Diese ist proportional zum Strom aus der Ionisationskammer. Wird die Strahlenschutz-Messung in einem gepulsten Strahlungsfeld durchgeführt, kann besonders bei kurzen Strahlungspulsen die Dosisleistung im Puls nicht mehr sicher abgelesen werden. In solchen Feldern bietet sich die Integration des Strompulses an. In diesem Fall wird die in der Ionisationskammer erzeugte Ladung gemessen. Die Ladung ist proportional zur im Strahlungsfeld applizierten Dosis. Aus der Dosis und der Pulslänge (falls bekannt) kann die Dosisleistung im Strahlungspuls nachträglich bestimmt werden. Da oft das Strahlungsfeld nicht genau bekannt ist bietet es sich an, beide Messungen (Dosisleistung und Dosis) parallel durchzuführen.

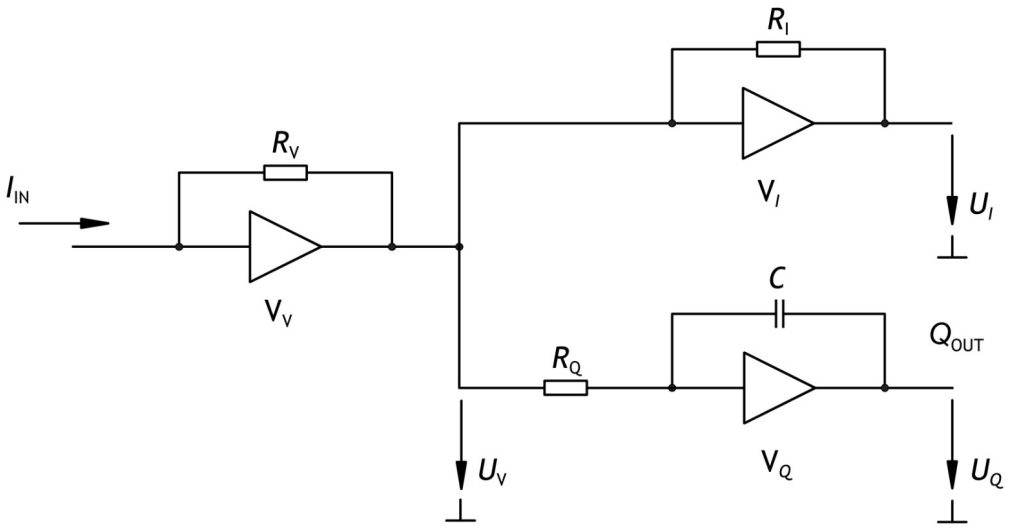


Abb. 1: Prinzipschaltbild des Puls-Elektrometer-Verstärkers.

Bei dem hier beschriebenen Puls-Elektrometer-Verstärker handelt es sich um einen für die oben beschriebene Messaufgabe entwickelten Prototyp. Im Kapitel 6, Ausblick, sind mögliche, aus den Erfahrungen mit dem Prototyp gewonnene Weiterentwicklungen beschrieben. Abb. 1 zeigt den prinzipiellen Aufbau des Verstärkers. Aus dem zu messenden Strom I_{IN} (10 fA < I_{IN} < 10 nA) wird mit dem Transimpedanz-Verstärker V_V eine niederohmig zu messende Spannung U_V erzeugt. In einer der beiden folgenden parallelen Verstärkerstufen wird diese Spannung weiter verstärkt. Der Ausgang dieses Verstärkerzweigs liefert die dem Eingangsstrom I_{IN} proportionale Spannung U_I. In dem zweiten Verstärkerzweig wird aus der Spannung U_V mit dem Widerstand R_Q ein Strom erzeugt. Dieser Strom wird mit dem Kondensator C integriert. Der Ausgang dieses Integrators liefert die der gemessenen Ladung ($I_{IN} * t$) proportionale Spannung U_Q. Abb. 2 zeigt den zeitlichen Verlauf von Strahlung, Strom und Ladung. Es ist zu sehen, dass zumindest die Dosis auch kurzer Strahlungsimpulse aus der integrierten Ladung abzulesen ist. Der zur Dosisleistung proportionale Strom kann nur in einem kontinuierlichen Strahlungsfeld eine Anzeige liefern, die die Dosisleistung am Messort sicher wiedergibt.

https://doi.org/10.7795/130.20210610

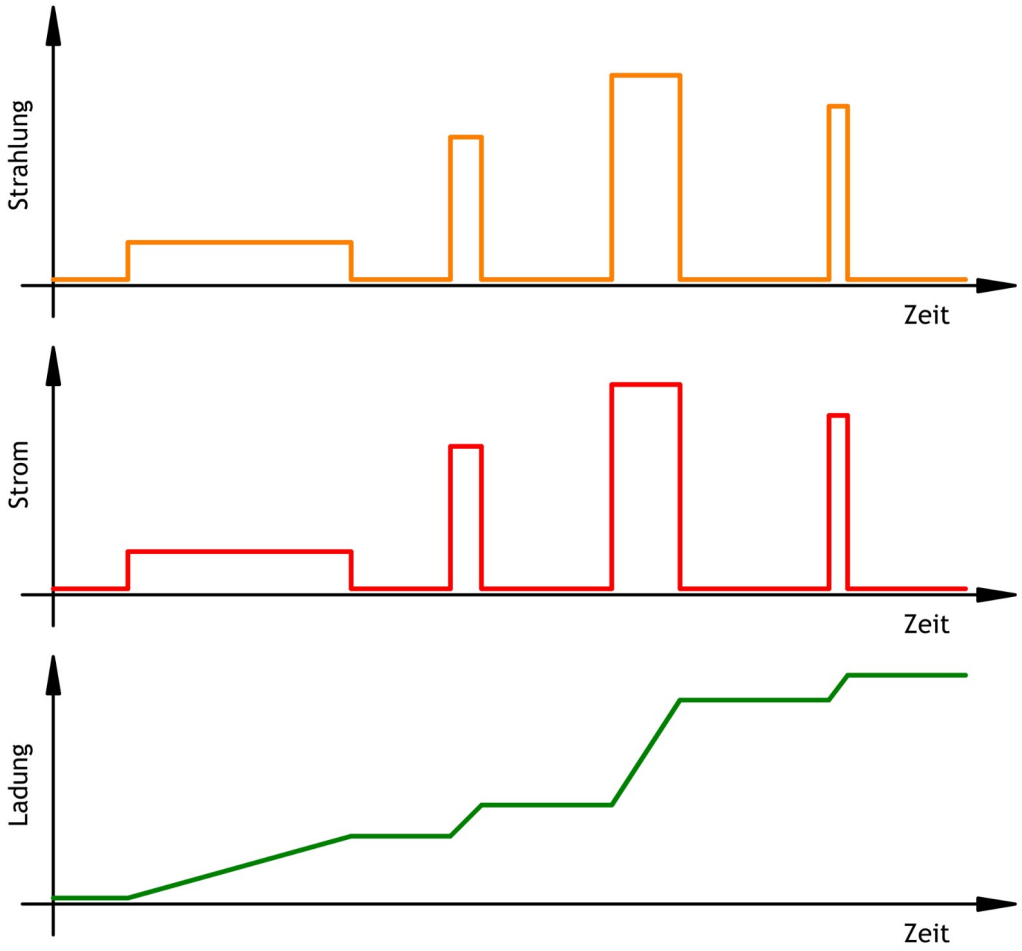


Abb. 3: Zeitlicher Verlauf von Strahlung, von gemessenem Strom und integrierter Ladung

In dem realisierten Aufbau des Puls-Elektrometer-Verstärkers kann die Verstärkung des Transimpedanz-Verstärkers mit dem Widerstand R_V um den Faktor 1000 umgeschaltet werden. Der Widerstand R_I kann ebenso wie die Kapazität C in drei Stufen um jeweils den Faktor 10 variiert werden. Somit ist die Messung von I_{IN} in insgesamt sechs Dekaden möglich.

https://doi.org/10.7795/130.20210610

2 Technische Realisierung

2.1 Generelle Anforderungen

Da der hier beschriebene Puls-Elektrometer-Verstärker in Verbindung mit einer Ionisations-Kammer in einem tragbaren Strahlenschutz-Dosimeter eingesetzt werden soll, sind einige zusätzliche Anforderungen bei der Realisierung einzuhalten. So benötigt die Kammer zum Betrieb eine zusätzliche Spannung zur Ladungstrennung. Je nach Aufbau der Kammer werden hierfür einige hundert Volt benötigt. Zusätzlich muss bei Messungen mit offenen Ionisationskammern die Luftdichte in der Kammer korrigiert werden. Hierzu sind der Luftdruck und die Temperatur möglichst in der Nähe der Kammer zu messen.

Für ein tragbares Strahlenschutz-Dosimeter kann als Stromversorgung nur ein Akku verwendet werden. Der Stromverbrauch von Verstärker und der nötigen Steuerung muss also möglichst gering sein. Zusätzlich ist die Überwachung des Akku – Ladezustands wichtig.

Besonders der Vorverstärker des Puls-Elektrometer-Verstärkers benötigt aufgrund der kleinen zu messenden Ströme eine eigene elektrische Schirmung.

2.2 Aufbau

Die für die oben beschriebenen Anforderungen benötigte Elektronik ist auf drei Platinen aufgeteilt. Auf einer Platine ist der Puls-Elektrometer-Verstärker aufgebaut. Eine weitere Platine ist mit dem Mikrocontroller zur Steuerung der Messung, dem Display, dem Luftdruck-Sensor und der Elektronik für den Temperatur-Sensor bestückt. Die Spannungsversorgung ist auf der dritten Platine mit dem Akku, den Spannungsreglern und dem Kammerspannungs-Modul aufgebaut. Die Platinen sind in einzelne, miteinander verbundene, Modulgehäuse eingebaut.

Abb. 4: Foto des fertig aufgebauten Puls-Elektrometer-Verstärkers
mit den Buchsen für Ionisationskammerstrom, Kammerspannung und Temperaturfühler

https://doi.org/10.7795/130.20210610

2.3 Puls-Elektrometer-Verstärker

2.3.1 Vorverstärker

Die Aufgabe des Vorverstärkers ist es, aus den kleinen Eingangsströmen gut weiterverarbeitbare Spannungen zu erzeugen. Der Vorverstärker ist als Transimpedanz-Verstärker aufgebaut. Abb. 5 zeigt die Schaltung des Vorverstärkers. Als Eingangs-Operationsverstärker wird der Baustein ADA4530-1 der Fa. ANALOG DEVICES (IC1) mit einem Eingangsruhestrom von etwa 0,1 fA verwendet. Da das Platinenmaterial für die kleinen zu messenden Ströme einen zu niedrigen Isolationswiderstand hat, sind alle mit dem Eingang elektrisch verbundenen Schaltungsteile frei durch die Luft verdrahtet und nicht über die Platine geführt. Um die thermische und elektrische Belastung des Eingangs-Verstärkers so klein wie möglich zu halten, wird dem Eingangs-Verstärker ein zweiter Operationsverstärker als Spannungsfolger in Reihe nachgeschaltet. Als Ausgangstreiber dient ein ADA4627 der Fa. ANALOG DEVICES (IC 3). In die Rückkopplung der beiden Verstärker ist ein 100 GΩ Widerstand (R2) geschaltet. Ein Eingangsstrom von 10 fA liefert mit diesem Rückkoppelwiderstand eine Ausgangsspannung von einem Millivolt. Zur Erweiterung des Messbereichs kann zu diesem Widerstand über das Relais K1 ein 100 MΩ Widerstand (R1) parallel geschaltet werden. Der nominelle Gesamtwiderstand dieser Parallelschaltung beträgt 99,9 MΩ. Mit diesen parallel geschalteten Widerständen in der Rückkopplung beträgt die Ausgangsspannung bei einem Eingangsstrom von 10 nA etwa ein Volt. Aus Strömen zwischen 10 fA und 10 nA können also mit diesem Vorverstärker gut messbare Spannungen erzeugt werden.

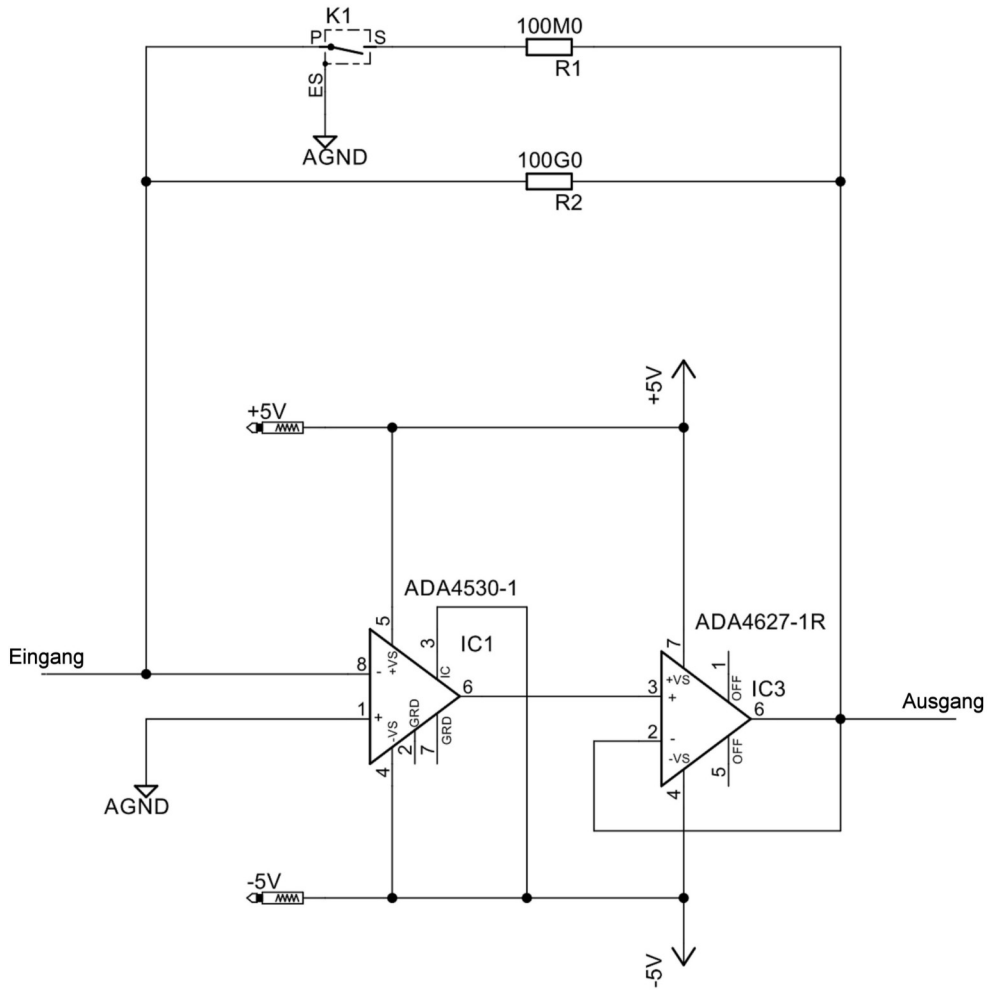


Abb. 5: Schaltung des Vorverstärkers

https://doi.org/10.7795/130.20210610

2.3.2 Strom-Verstärker

Der zur Spannungsmessung verwendete ADC hat einen Messbereich von 0 V bis 5 V. Um diesen Bereich mit möglichst guter Auflösung auszunutzen zu können, muss die Ausgangsspannung des Vorverstärkers weiter verstärkt werden.

Für die Stromanzeige geschieht dies mit zwei weiteren Operationsverstärkern. Die Verstärkung beider Verstärker kann mit den Relais K2 und K3 jeweils zwischen dem Faktor 1 und 10 umgeschaltet werden. Insgesamt sind also die Verstärkungsstufen 1, 10 und 100 möglich. Bei einem Eingangsstrom von 10 fA stehen für den ADC mit der weiteren Verstärkung um den Faktor 100 somit 100 mV als kleinste zu messende Spannung zur Verfügung.

Der Vorverstärker invertiert das Eingangssignal. Damit das Ausgangssignal die Polarität des Eingangssignals behält, erfolgt in dieser Verstärkerstufe eine weitere Invertierung. Dies wird dadurch erreicht, dass ein Operationsverstärker als invertierender und einer als nicht invertierender Verstärker beschaltet ist. Der nicht invertierende Verstärker wird wegen seines höheren Eingangswiderstands als Eingangs-Verstärker benutzt.

Als Operationsverstärker werden zwei ADA4627 der Fa. ANALOG DEVICES (IC5 und IC2) verwendet. Abb. 6 zeigt die Schaltung des Strom-Verstärkers.

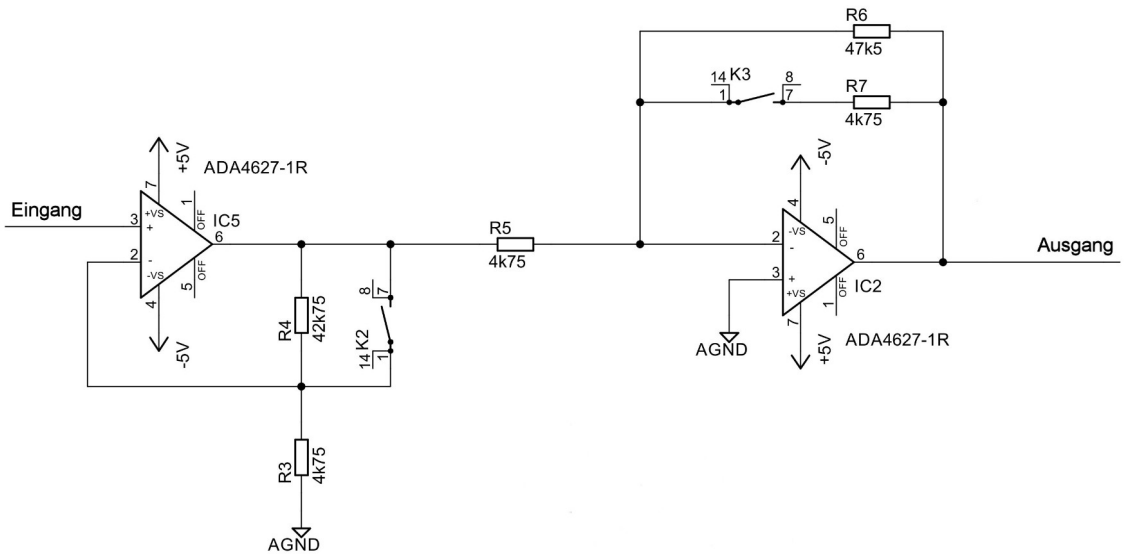


Abb. 6: Schaltung des Strom-Verstärkers

2.3.3 Ladungs-Verstärker

Die Ladung wird mit einem analogen Integrator gemessen. Zwar könnte die Ladung auch digital durch Aufsummieren des Eingangsstroms gemessen werden, allerdings können kurze Stromimpulse aufgrund der für die Analog/Digital-Wandlung benötigten Zeiten unter Umständen nicht erkannt werden. Bei der analogen Integration wird die Ladung auf einem Kondensator gesammelt und die Spannung über dem Kondensator ohne zeitliche Probleme gemessen.

Zur analogen Messung der Ladung muss aus der Ausgangsspannung des Vorverstärkers wieder ein Strom erzeugt werden. Dies geschieht mit dem 10 MΩ Widerstand R8. Eine Eingangsspannung von 10 mV liefert für den Integrator einen gut zu messenden Eingangsstrom von 1 nA. Dieser Strom liefert über einem 10 nF Kondensator nach 10 s Integrationszeit am Ausgang eine Spannung von einem Volt.

https://doi.org/10.7795/130.20210610

Der Eingangsstrom wird auf einem von drei möglichen Kondensatoren gesammelt. Über die Relais K4 und K5 kann zu dem fest in die Rückkopplung des Verstärkers geschalteten Kondensator C1 mit einer Kapazität von 10 nF jeweils ein um Faktor 10 bzw. Faktor 100 größerer Kondensator parallel geschaltet werden. Zum Entladen werden die Kondensatoren über das Relais K6 und den Widerstand R9 kurzgeschlossen.

Als Operationsverstärker wird auch hier ein ADA4627 der Fa. ANALOG DEVICES (IC4) verwendet. Abb. 7 zeigt die Schaltung des Ladungs-Verstärkers.

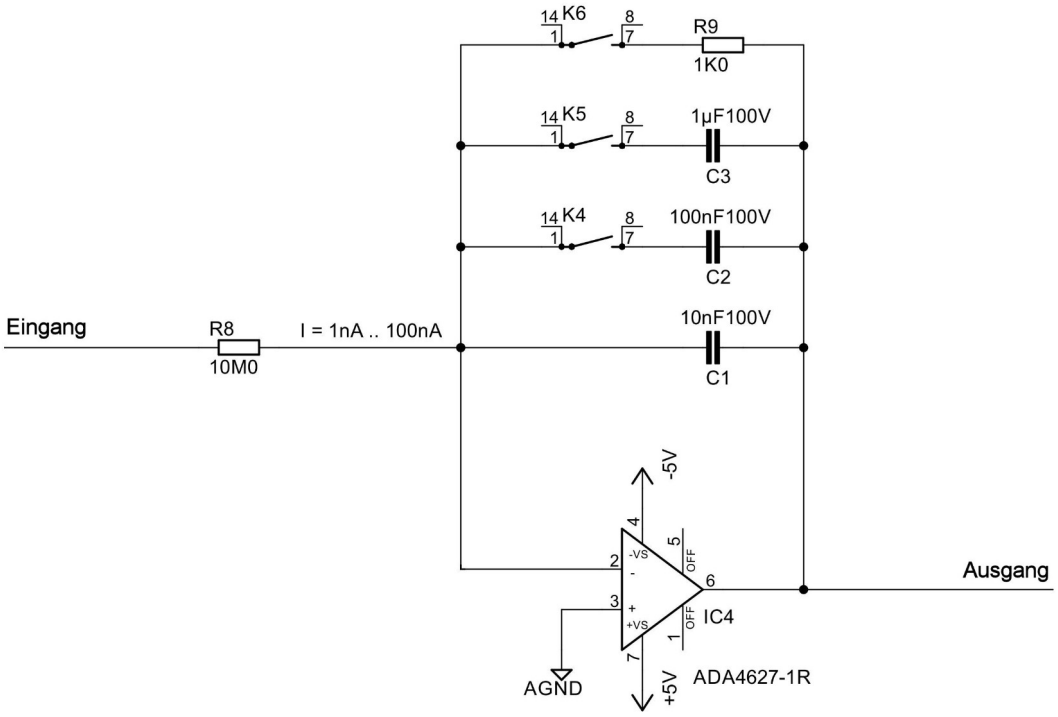


Abb. 7: Schaltung des Ladungs-Verstärkers

2.4 Steuerung

Zur Steuerung und Anzeige der Messwerte des Puls-Elektrometer-Verstärkers wird der Mikrocontroller eines Arduino Nanos verwendet. Abb. 8 zeigt den prinzipiellen Aufbau der Elektrometer-Steuerung.

Die Relais zur Einstellung der Elektrometer-Messbereiche werden über die I/O-Leitungen des Mikrocontrollers geschaltet.

Die Messung der zum Strom und zur Ladung proportionalen Ausgangsspannungen des Puls-Elektrometer-Verstärkers erfolgt mit dem Analog/Digital-Wandler (ADC) des Mikrocontrollers.

Der Puls-Elektrometer-Verstärker soll zur Messung von Dosis bzw. Dosisleistung mit einer Ionisationskammer eingesetzt werden. Zur Korrektur der Luftdichte in der Kammer müssen Luftdruck und Temperatur gemessen werden. Hierzu wird für den Luftdruck ein BME280-Sensor eingesetzt. Der Sensor wird über eine digitale I2C-Schnittstelle ausgelesen. Da die Temperatur der Luft in der Nähe der Ionisationskammer gemessen werden soll, wird ein an die Steuerung anzuschließender PT100-Fühler verwendet. Die Messung der Temperatur erfolgt mit dem Widerstand-zu-Digital-Konverter MAX31865 der Fa. Maxim. Der Anschluss des Konverters an den Mikrocontroller erfolgt über die SPI-Schnittstelle. Um den Aufbau der Steuerungsplatine zu vereinfachen, werden für

https://doi.org/10.7795/130.20210610

Luftdrucksensor und Widerstand-zu-Digital-Konverter fertig bestückte Breakout Boards der Firma Adafruit verwendet.

Die Elektrometer-Messbereiche werden über ein Touchdisplay eingestellt und der eingestellte Bereich sowie die auf Luftdruck und Temperatur korrigierten Messwerte von Dosis und Dosisleistung werden auf dem Display angezeigt. Zum Betrieb der Ionisationskammer wird eine Kammerspannung von einigen hundert Volt benötigt. Diese Spannung wird zusammen mit dem Luftdruck und der Temperatur auf dem Display angezeigt. Der Anschluss des Displays an den Mikrocontroller erfolgt über die SPI-Schnittstelle des Mikrocontrollers.

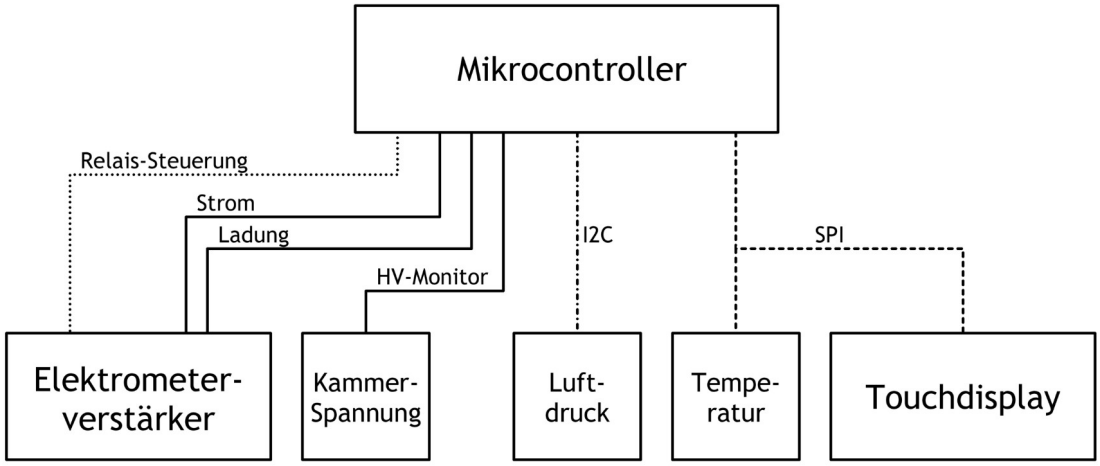


Abb. 8: Prinzip der Elektrometer-Steuerung

2.4.1 Spannungs-Messung

Zur Messung der Elektrometer-Ausgangsspannungen sowie der HV-Monitorspannung und der Akkuspannung werden die Analogeingänge des Mikrocontrollers benutzt. Der im Arduino Nano verwendete ATmega328 stellt acht Analogeingänge mit einer Auflösung von 10 Bit zur Verfügung. Die maximale Eingangsspannung beträgt 5 V. Eine zu messende Spannung kann also nur mit einer maximalen Auflösung von etwa 5 mV gemessen werden.

Zum Abgleich des Analog/Digital-Wandlers (ADC) werden Nullpunkt (Offset) und Endwert (Verstärkung) des ADC bestimmt. Eine durch beide Werte gelegte Gerade wird zur Korrektion der vom ADC ausgegebenen Spannung benutzt.

Gerade bei kleinen Elektrometer Ausgangsspannungen führt die geringe Auflösung des ADCs zu starken Schwankungen in der Anzeige. Ein weiteres Problem sind einzelne Spannungsmesswerte, die deutlich vom Wert der angelegte Spannung abweichen. Um eine möglichst genaue und stabile Anzeige zu erhalten, werden zwei Filterverfahren angewendet. Zuerst wird aus fünf nacheinander gemessenen Spannungswerten der Median bestimmt. Dieser Filter eliminiert einzelne Spitzenwerte. Die Median-gefilterten Messwerte werden danach über eine Sekunde gemittelt und auf dem Display ausgegeben.

2.5 Versorgung

Für die Versorgung von Elektrometer und Steuerung werden verschiedene Betriebsspannungen benötigt. Diese Spannungen werden mit Gleichspannungswandlern (DC/DC – Wandler) aus einem 7,4 V Lithium-Ionen-Akkumulator erzeugt. Für die Steuerung stehen so 5 V und für den Analogteil ±12 V zur Verfügung. Die verwendeten DC/DC-Wandler sind potentialfrei, so dass die Massen von Steuerspannung (GND) und Analogspannung (AGND)

https://doi.org/10.7795/130.20210610

zur Vermeidung von Störungen getrennt gehalten werden können. Um eine Tiefentladung des Akkus zu verhindern, wird die Akku-Spannung mit einer Schutzschaltung überwacht. Unterschreitet die Akku-Spannung einen Minimalwert, werden die DC/DC-Wandler vom Akku getrennt. Um den Benutzer rechtzeitig vor dem Abschalten zu warnen, wird über einen Analogeingang des Mikrocontrollers die Akku-Spannung gemessen. Bei einer zu niedrigen Akku-Spannung wechselt die Farbe der Anzeige von Luftdruck, Temperatur und Kammerspannung von Weiß zu Rot.

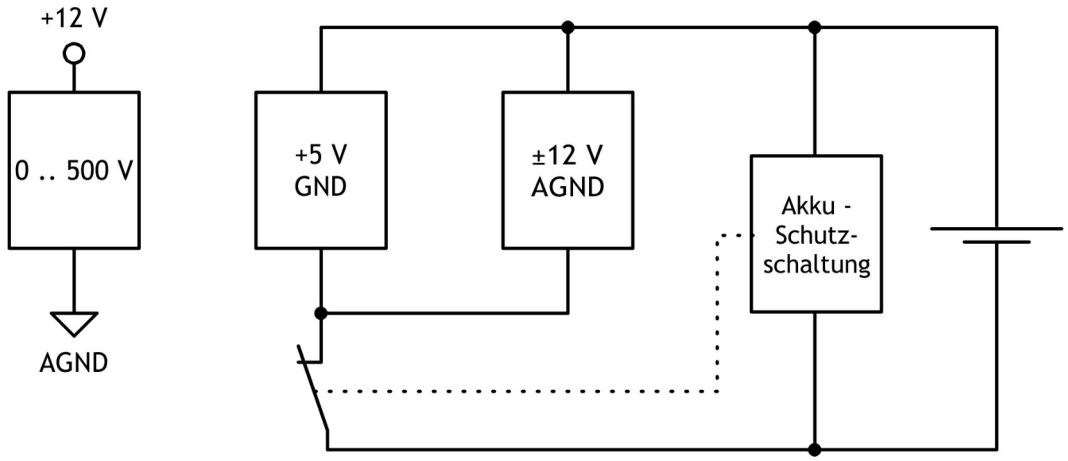


Abb. 9: Aufbau der Spannungsversorgung

Die für den Betrieb einer Ionisationskammer benötigte Kammerspannung wird mit einem Hochspannungsmodul aus den +12 V der Versorgung des Analogteils erzeugt. Abb. 9 zeigt den prinzipiellen Aufbau der Spannungsversorgung.

2.5.1 Akkubetrieb

Für die Spannungsversorgung wird ein aus zwei Samsung ICR18650 Lithium-Ionen-Akkus bestehender Akkupack verwendet. Seine Ausgangsspannung beträgt 7,4 V und seine Kapazität 2600 mAh. Geladen werden die Akkus mit einem Ansmann Ladegerät für zwei Lithium-Ionen-Zellen (LBC-2cells LiIon). Das Ladegerät liefert einen Strom von 2 A. Leere Akkus sind damit in knapp 3 Stunden voll aufgeladen.

2.5.2 Kammerspannung

Zur Erzeugung der Kammerspannung wird ein Hochspannungs DC/DC- Wandler der Firma hivolt mit der Bezeichnung HM34S-0.5P-12 eingesetzt. Der Wandler erzeugt aus einer 12 V Eingangsspannung eine Hochspannung von 500 V. Der Hersteller gibt die Stabilität der Hochspannung mit 100 ppm/h und den Temperaturkoeffizienten mit 25 ppm/K an.

Die Ausgangsspannung kann über eine Steuerspannung (0 bis 5 V) eingestellt werden. Der Hochspannungs DC/DC- Wandler stellte eine Referenzspannung von 5 V zur Verfügung. Die Referenzspannung wird über einen mit einem Stufenschalter umschaltbaren Spannungsteiler als Steuerspannung benutzt. Die Ausgangsspannung kann so in Stufen von 100 V von 0 bis 500 V eingestellt werden. Die eingestellte Spannung kann über einen Hochspannungsmonitor kontrolliert werden. Der Monitorausgang liefert eine Spannung von 0 bis 5 V. Diese Spannung wird über einen Analogeingang des Mikrocontrollers gemessen und umgerechnet (mit 100 multipliziert) im Display der Steuerung angezeigt.

https://doi.org/10.7795/130.20210610

3 Bedienung

Der Puls-Elektrometer-Verstärker wurde zum Betrieb mit einer offenen Ionisationskammer entwickelt. Vor einer Messung müssen der Stromausgang und der Kammerspannungseingang der Kammer mit dem Verstärker verbunden werden. Außerdem muss ein PT100 Temperaturfühler angeschlossen werden.

Die Bedienung des Puls-Elektrometer-Verstärkers erfolgt über das Touch-Display. Abb. 10 zeigt die anzeigten Werte und die Bedienfelder.

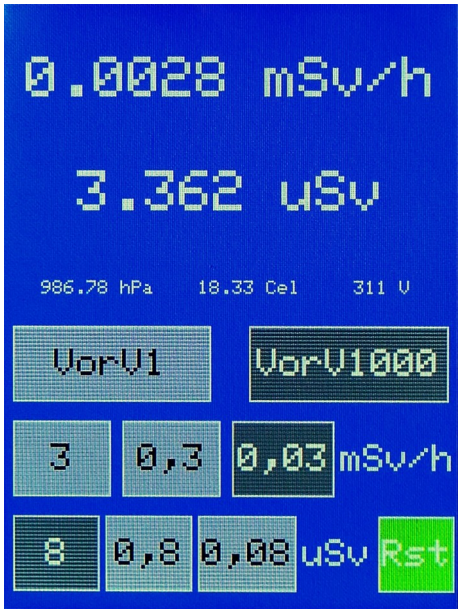


Abb. 10: Touch-Display mit Anzeige und per Touch auszuwählenden Messbereichen.

Die beiden oberen Zeilen der Anzeige zeigen die in der letzten Sekunde gemessenen Werte für Dosisleistung und Dosis. In der dritten Zeile werden Luftdruck, Temperatur und der Monitorwert der Kammerspannung angezeigt. Sinkt die Spannung des Lithium-Ionen-Akkumulators der Spannungsversorgung unter 6,1 V, werden die drei Werte in dieser Zeile in rot dargestellt. Das Gerät wird wie in 2.5 beschrieben, bei einer Akku-Spannung von etwa 5,9 V abgeschaltet.

Die Einstellung des gewünschten Messbereichs erfolgt über die Schaltflächen in den nächsten drei Zeilen. Die aktuell eingestellten Bereiche werden jeweils dunkel dargestellt. Mit den mit „VorV1“ bzw. „VorV1000“ beschrifteten Schaltflächen wird der Verstärkungsfaktor des Vorverstärkers gewählt. „VorV1“ schaltet die Verstärkung auf 1, „VorV1000“ auf 1000. Mit der gewählten Vorverstärkung ändern sich die Messbereiche für Dosis und Dosisleistung. Die Endwerte der zur Verfügung stehenden Bereiche werden auf den Schaltflächen dargestellt. Eine Dosis-Messung wird durch Betätigung der Schaltfläche des gewünschten Bereichs gestartet. Die „Rst“-Schaltfläche wird in grün dargestellt. Die gesammelte Dosis kann mit der „Rst“-Schaltfläche auf null zurückgesetzt werden. Die „Rst“-Schaltfläche wird jetzt in rot dargestellt. Zum erneuten Start der Dosis-Messung ist die Schaltfläche des gewünschten Bereichs erneut zu betätigen.

https://doi.org/10.7795/130.20210610

3.1.1 Ausgabe über RS232

Neben den Elektrometer- und den Klima-Messwerten werden die Zeit nach dem Einschalten des Elektrometers und die Batteriespannung über den USB-Port des Mikrocontrollers ausgegeben.

In einem angeschlossenen Windows-Rechner erscheint die Schnittstelle als COM-Port. Für den COM-Port sind die in Abb. 11 gezeigten Einstellungen vorzunehmen.

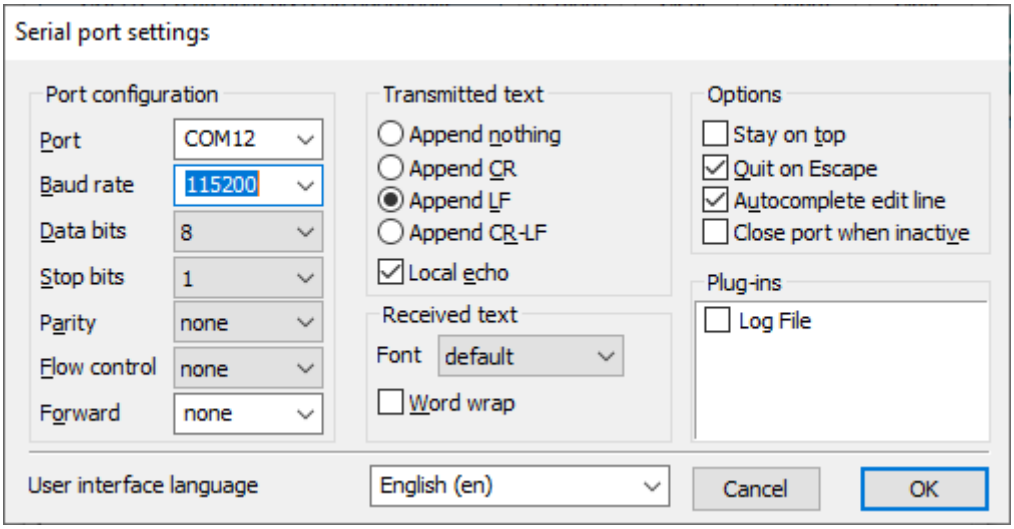


Abb. 11: COM-Port Einstellungen des Terminal-Programms Termite als Beispiel

Die Daten werden im folgenden Format übertragen:

Zeit in s	Dosisleist. in mSv/h	Dosis in µSv	Strom in pA	Ladung in pC	Luftdruck in hPa	Temperatur in °C	Kammerspg. in V	Batteriespg. in V
332 ;	0,0252 ;	0,303 ;	0,203 ;	0,321 ;	1001,73 ;	19,89 ;	300 ;	8,24 ;
333 ;	0,0250 ;	0,303 ;	0,201 ;	0,321 ;	1001,73 ;	19,89 ;	300 ;	8,24 ;
334 ;	0,0251 ;	0,303 ;	0,200 ;	0,321 ;	1001,73 ;	19,90 ;	300 ;	8,23 ;

Die einzelnen Messwerte in einer Zeile werden jeweils durch ein Semikolon getrennt.

Die Daten können mit einem Terminal-Programm wie z.B. Termite aufgezeichnet werden.

https://doi.org/10.7795/130.20210610

4 Kalibrierfaktoren und wichtige Konstanten

Unter der Überschrift „define: Strom- und Ladungs-Messung“ sind im Hauptprogramm (Kapitel 8.2.1) des Mikrocontrollers die für die Ermittlung der Messwerte wichtigen Konstanten zusammengefasst. Die Konstanten-Namen sind im folgenden Text *kursiv* dargestellt.

So enthält *u_plus4V9* den Ausgabewert des ADC bei einer Eingangsspannung von 4,9 V. Entsprechend enthält *u_plus0V1* den Ausgabewert des ADC bei einer Eingangsspannung von 0,1 V. Die beiden Werte werden wie in 2.4.1 beschrieben zur Korrektion der ADC-Spannungs-Messung benutzt.

Wie in 2.4.1 beschrieben, wird jeder Spannungsmesswert über eine Anzahl von Einzelmessungen gemittelt. Diese Anzahl wird mit der Konstanten *AnzMittelwertAnalog* festgelegt.

Die maximal erlaubten Ausgangsspannungen, jeweils in Volt, des Strom- und des Ladungs-Verstärkers sind in den Konstanten *MaxAusgSpgStrom* bzw. *MaxAusgSpgLadung* festgelegt.

Die Kalibrierfaktoren der Strom- und Ladungs-Messbereiche werden in sechs Faktoren für die Strom-Messung und sechs Faktoren für die Ladungs-Messung gespeichert. Die Konstanten-Namen bestehen aus der gewählten Vorverstärkung (VV1000 oder VV1) und der Verstärkung des Ladungs- bzw. Strom-Verstärkers (VQ oder VI). Die Konstante *VV1000_VI10* z.B. enthält also den Kalibrierfaktor für den Strom-Messbereich mit dem Verstärkungsfaktor 10 und der Vorverstärkung 1000. *VV1_VQ100* enthält den Kalibrierfaktor für den Ladungsmessbereich mit dem Verstärkungsfaktor 100 und der Vorverstärkung 1.

Der Kammerfaktor der anzuschließenden Ionisationskammer ist in der Konstanten *KammerFaktor_mSv_C* in mSv/C gespeichert.

Die drei den zeitlichen Ablauf der Messung steuernden Timer werden mit Konstanten eingestellt. Die Inhalte der Konstanten geben jeweils eine Zeit in Millisekunden an. Im *IntervalAnalog* werden Messungen, wie in 2.4.1 beschrieben, mit dem ADC durchgeführt. Hierzu wird die function AnalogLesen aufgerufen. Die function AnalogLesen benötigt etwa 6 ms um einen Messwert zu liefern. Werte für *IntervalAnalog* müssen also größer als 6 sein. Im *IntervalButton* wird abgefragt ob der Benutzer ein Feld auf dem Touch-Display betätigt hat. Die Konstante *IntervalPrint* gibt an, in welchem Zeitintervall Werte auf dem Touch-Display und über den USB-Port ausgegeben werden.

Die Konstante *UBattLow* enthält die Akku-Spannung, ab der die Anzeige der Klima-Messwerte und der Kammerspannung von weiß auf rot umgeschaltet wird.

https://doi.org/10.7795/130.20210610

5 Test-Messungen

Da es sich bei dem hier beschriebenen Puls-Elektrometer-Verstärker um einen Prototyp handelt, wurden nur Testmessungen durchgeführt, um das Messprinzip auf seine Eignung für Strahlenschutzmessungen zu überprüfen.

Eine wichtige Eigenschaft von Strahlenschutzmessgeräten ist die untere Messbereichsgrenze. Ein Kriterium hierfür ist der Eingangsruhestrom des Elektrometerverstärkers. Abb. 12 zeigt den zeitlichen Verlauf des Eingangsruhestroms im kleinsten Messbereich ohne Eingangssignal.

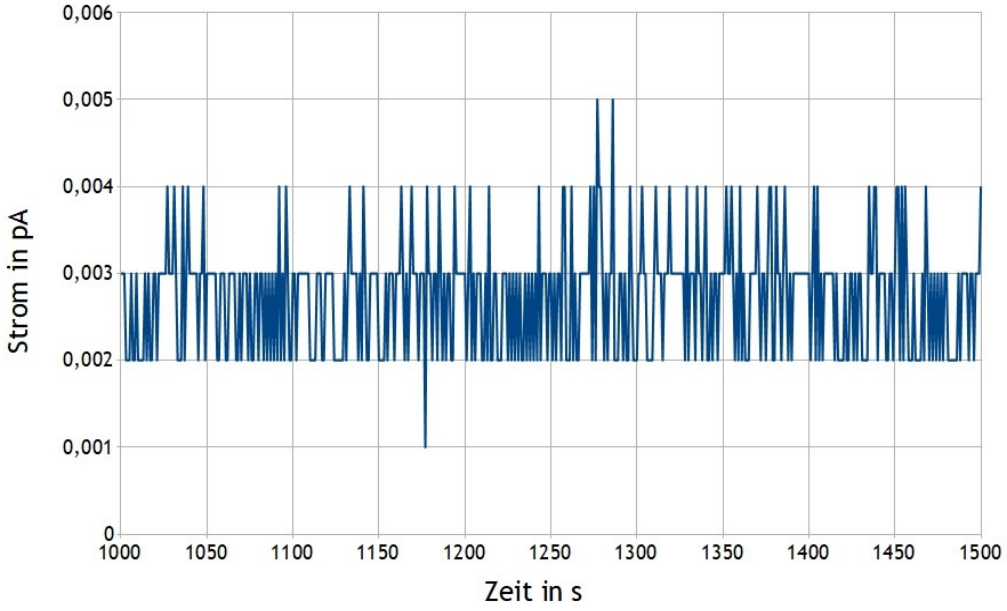


Abb. 12: Eingangsruhestrom im kleinsten Messbereich ohne Eingangssignal

Die Ionisationskammer und die Kabelverbindungen können die Eigenschaften des Strahlenschutzmessgeräts wesentlich verschlechtern. Abb. 13 zeigt eine Messung mit einer Ionisationskammer mit einem Messvolumen von 1 l in einem Strahlenfeld bei einer Dosisleistung von 1 µSv/h.

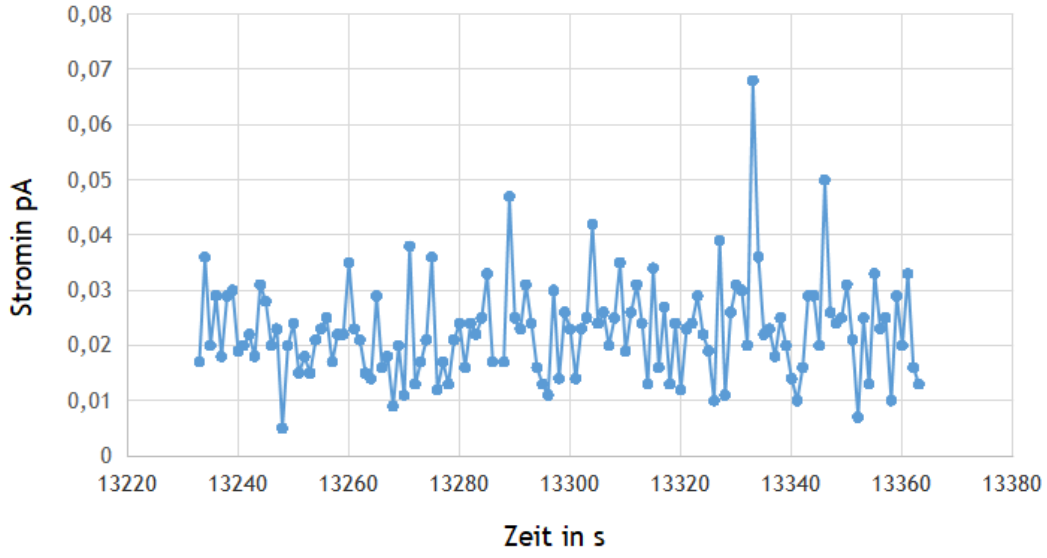


Abb. 13: Messung mit einer Ionisationskammer bei einer Dosisleistung von 1 µSv/h

https://doi.org/10.7795/130.20210610

Eine Messung mit einem vergleichbaren Eingangsstrom aus einer elektrischen Stromquelle und einer für diese Ströme geeigneten Eingangsleitung zeigt, dass wesentlich kleinere Unsicherheiten erreichbar sind. Abb. 14 zeigt die Messung eines Eingangsstroms von 5 fA im Zeitraum von 1760 s bis 1805 s. Die Strom-Peaks entstehen als Störsignale beim Umschalten der Stromquelle von 0 fA auf 5 fA.

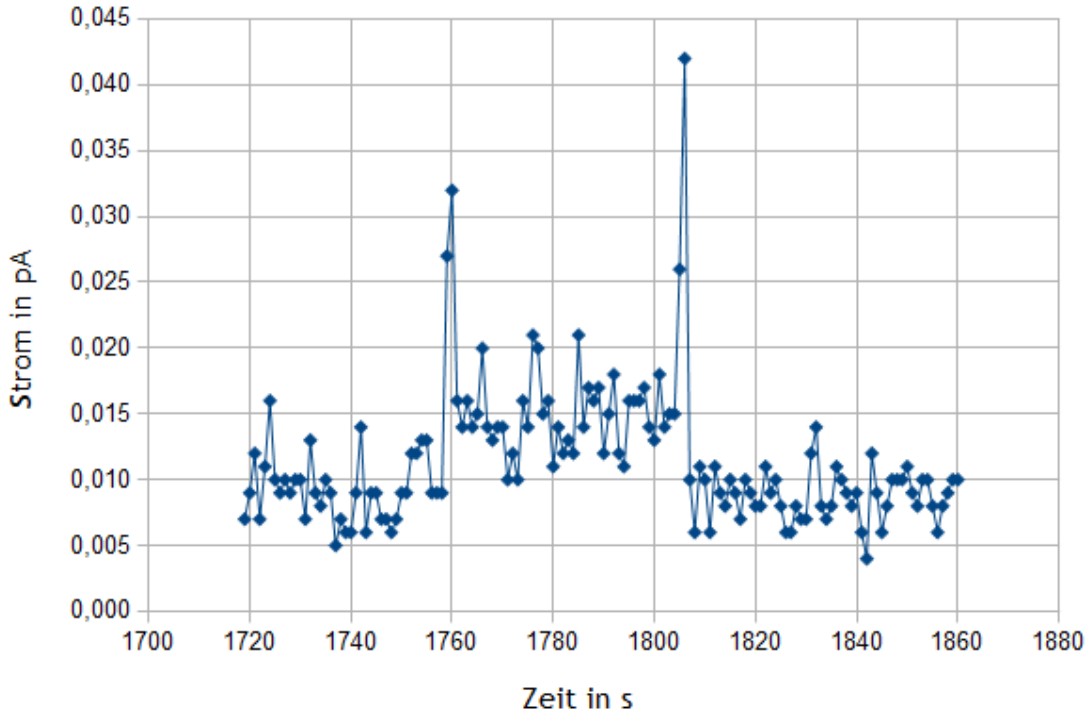


Abb. 14: Messung eines Eingangsstroms von 5 fA aus einer elektrischen Stromquelle. Die Peaks sind Störungen, die beim Umschalten der Stromquelle entstehen.

https://doi.org/10.7795/130.20210610

Ein weiteres Kriterium ist die Linearität über den gesamten Messbereich. Abb. 15 zeigt das Verhältnis vom Messwert bezogen auf den richtigen Wert der Messgröße. Die systematische Abweichung von der zu erwartenden 1 beruht auf der Energieabhängigkeit der verwendeten Ionisationskammer. Das Messgerät verwendet einen mittleren Kalibrierfaktor für eine Photonenenergie von 250 keV. Die Bestrahlungen wurden mit Cs-137 mit einer Photonenenergie von 662 keV durchgeführt. Das Verhältnis der Kalibrierfaktoren der Ionisationskammer bei den beiden Energien ist 1,09.

Die großen Abweichungen von 1 in dem kleinsten Messbereich beruhen auf den großen Unsicherheiten bei der Messung (siehe Abb. 13).

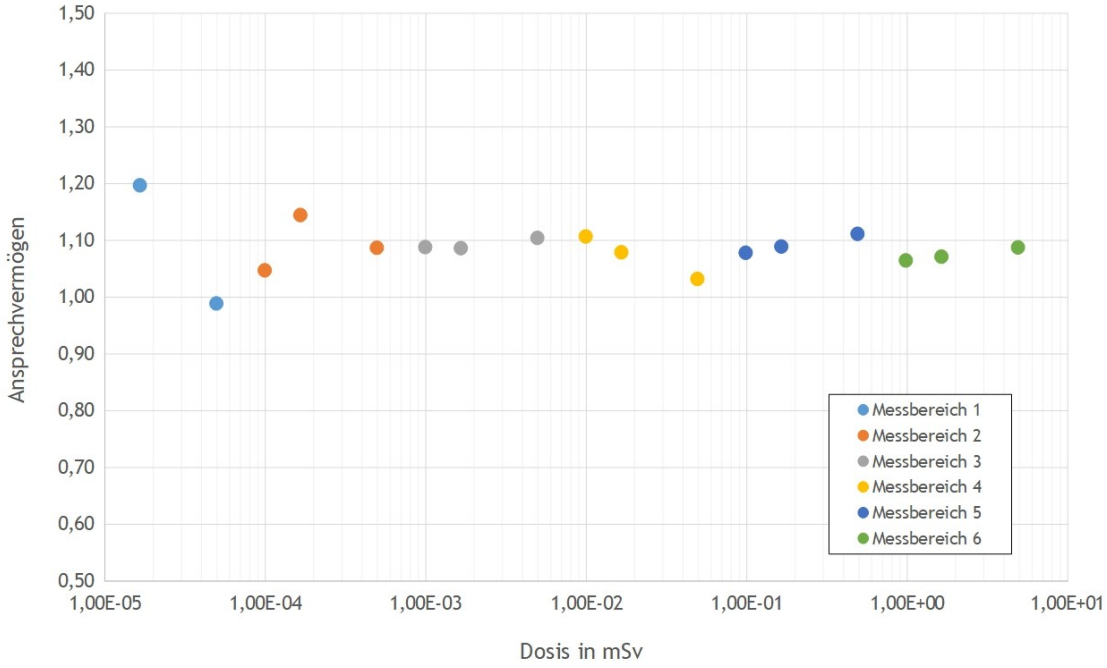


Abb. 15: Linearität der Dosisanzeige bzw. der Ladungsmessung

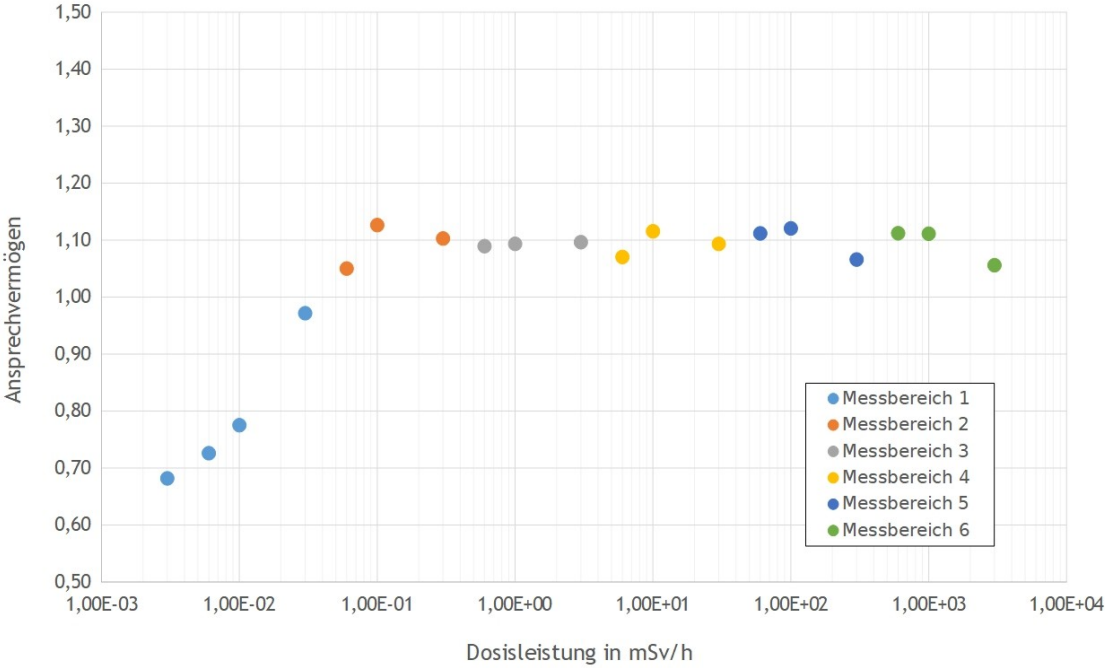


Abb. 16: Linearität der Dosisleistungsanzeige bzw. der Strommessung

https://doi.org/10.7795/130.20210610

6 Ausblick

Die Erfahrungen mit dem hier beschriebenen Elektrometer haben gezeigt, dass einige Verbesserungen an der Elektronik und der Software möglich sind.

Da der Vorverstärker nur durch die hochohmigen Strom- und Ladungs-Verstärker belastet wird, kann hier auf den in 2.3 beschriebenen Spannungsfolger verzichtet werden. Jeder Operationsverstärker fügt dem Signal unerwünschtes Rauschen hinzu. Dies kann durch diese Maßnahme reduziert werden.

Der verwendete Eingangs-Operationsverstärker stellt einen Guard-Ausgang zur Verfügung. Mit diesem niederohmigen Ausgang kann der Schirm des Eingangssignals auf dem gleichen Potential wie der Eingang selbst gehalten werden. Von der Offsetspannung des Eingangs durch den Isolationswiderstand des Eingangskabels getriebene Leckströme werden so verhindert.

In einer Weiterentwicklung sollten die beiden oben beschriebenen Verbesserungen berücksichtigt werden.

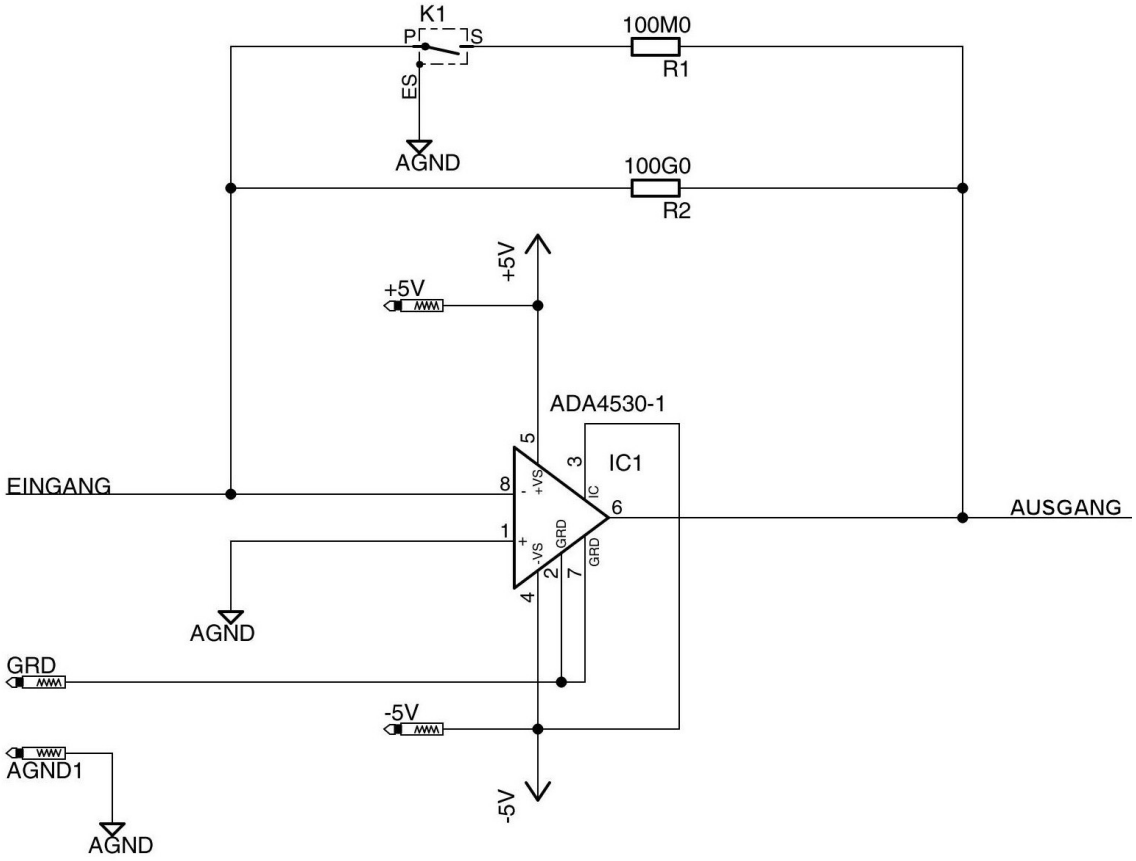


Abb. 17: Schaltung des modifizierten Vorverstärkers

Da in gepulsten Strahlungsfeldern vor der Messung die Dosis im Puls nicht bekannt ist, wurde bei den Test-Messungen oft ein nicht optimaler Messbereich gewählt. Dies hatte zur Folge, dass Messungen oft wiederholt werden mussten.

Wie in 2.3 beschrieben, wird bei diesem Elektrometer der Strom für die Ladungs-Messung mit einem Vorwiderstand aus dem niederohmigen Ausgangssignal des Vorverstärkers erzeugt. Es ist darum möglich, drei Ladungs-Verstärker mit unterschiedlichen Rückkoppelkapazitäten parallel zu betreiben. Jeder Ladungs-Verstärker benötigt lediglich seinen eignen Vorwiderstand. Die zusätzliche Belastung des Vorverstärkers ist aufgrund der hohen Vorwiderstände gering. Die Steuerung kann so aus den drei parallel zur Verfügung stehenden Ladungsmesswerten den optimalen auswählen.

https://doi.org/10.7795/130.20210610

Bei diesem Aufbau des Ladungs-Verstärkers entfallen die Relais für die Messbereichsumschaltung, dafür muss für jeden Ladungskanal ein eigenes Reset-Relais vorhanden sein. Da die Spulen dieser Relais einen hohen Anteil am Stromverbrauch haben ist zu prüfen, ob statt der Relais Analogschalter eingesetzt werden können. Es sind Analogschalter verfügbar, die ähnliche Isolationswiderstände wie die der verwendeten Reed-Relais haben. Diese Analogschalter können natürlich auch in der Messbereichsumschaltung des Strom-Verstärkers eingesetzt werden.

Abb. 18 zeigt die Schaltung des modifizierten Ladungs-Verstärkers.

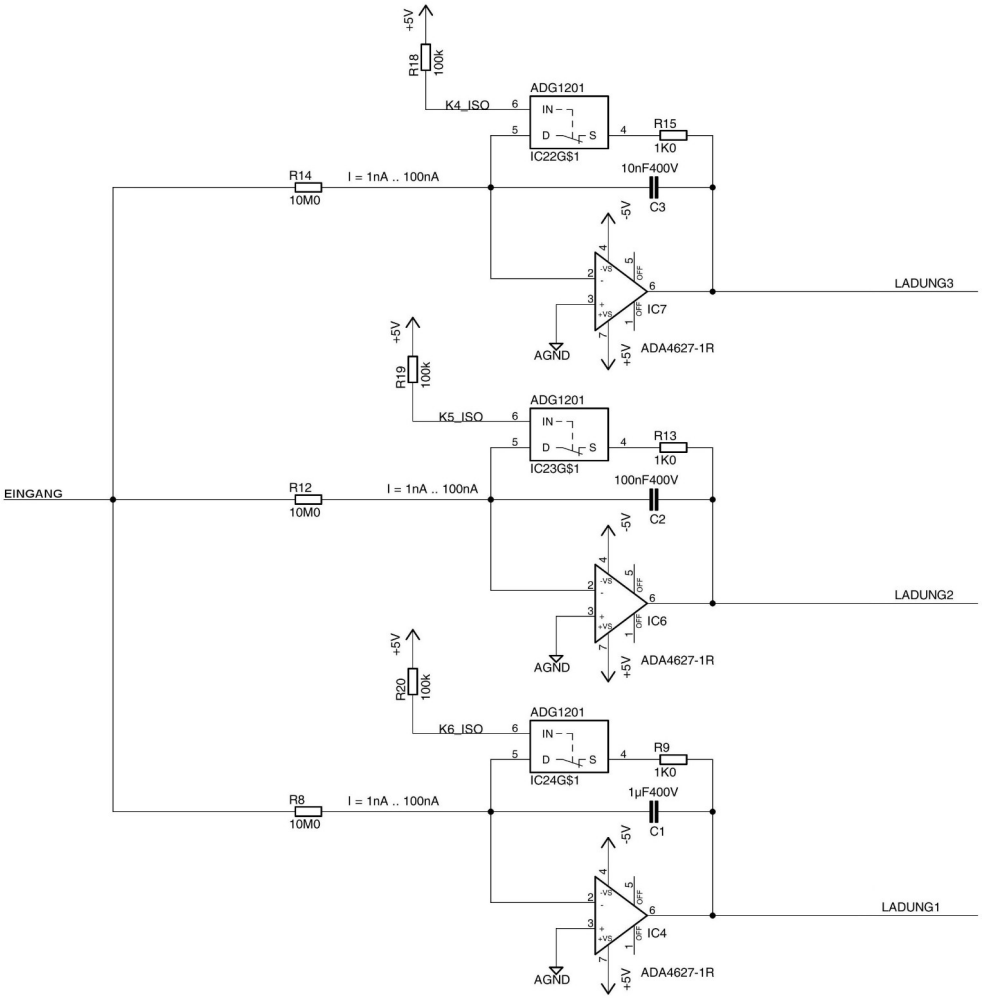


Abb. 18: Schaltung des modifizierten Ladungs-Verstärkers

Ein weiterer Schwachpunkt des hier beschriebenen Prototyps ist der verwendete Mikrocontroller. Zum einen verfügt er über einen zu kleinen Programmspeicher, der im Wesentlichen für das Touch-Display benötigt wird. Zum anderen ist der ADC des Mikrocontrollers mit seiner 10 Bit Auflösung ein Schwachpunkt. Wünschenswerte Software-Erweiterungen, wie z.B. eine Fernsteuerung, sind somit nicht möglich. Es sind eine Reihe von leistungsstärkeren Mikrocontrollern erhältlich, so dass mit ihnen zusätzliche Optionen realisiert werden können.

7 Danksagung

Herrn Fuhg danke ich für die Durchführung der aufwändigen Testmessungen sowie für die umfangreiche Auswertung der Messergebnisse.

Herrn Winterbottom danke ich für die Konstruktion und den Bau der mechanischen Komponenten des Puls-Elektrometer-Verstärkers.

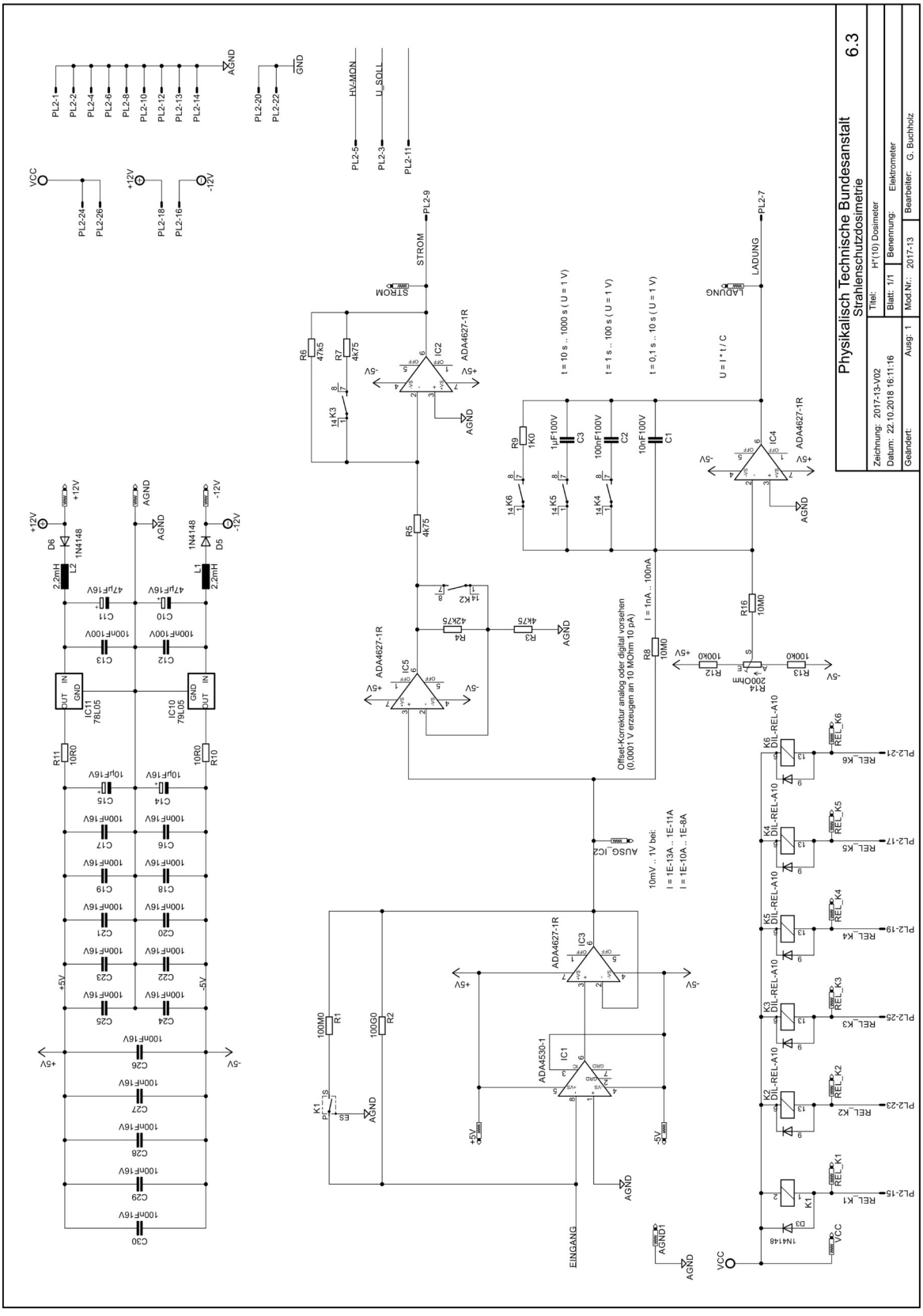
https://doi.org/10.7795/130.20210610

8 Anhang

8.1 Schaltpläne und Platinen-Layouts

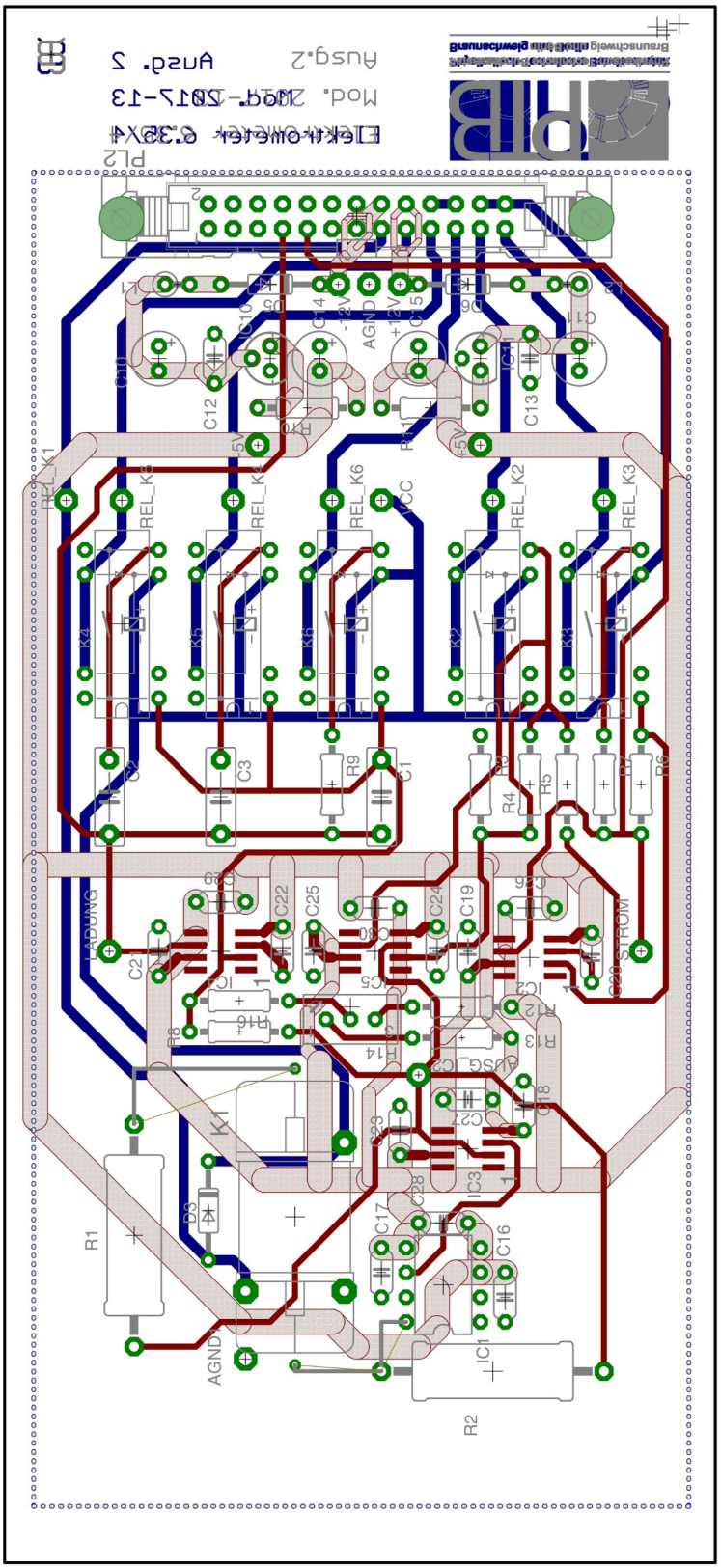
8.1.1 Puls-Elektrometer-Verstärker

Schaltplan



https://doi.org/10.7795/130.20210610

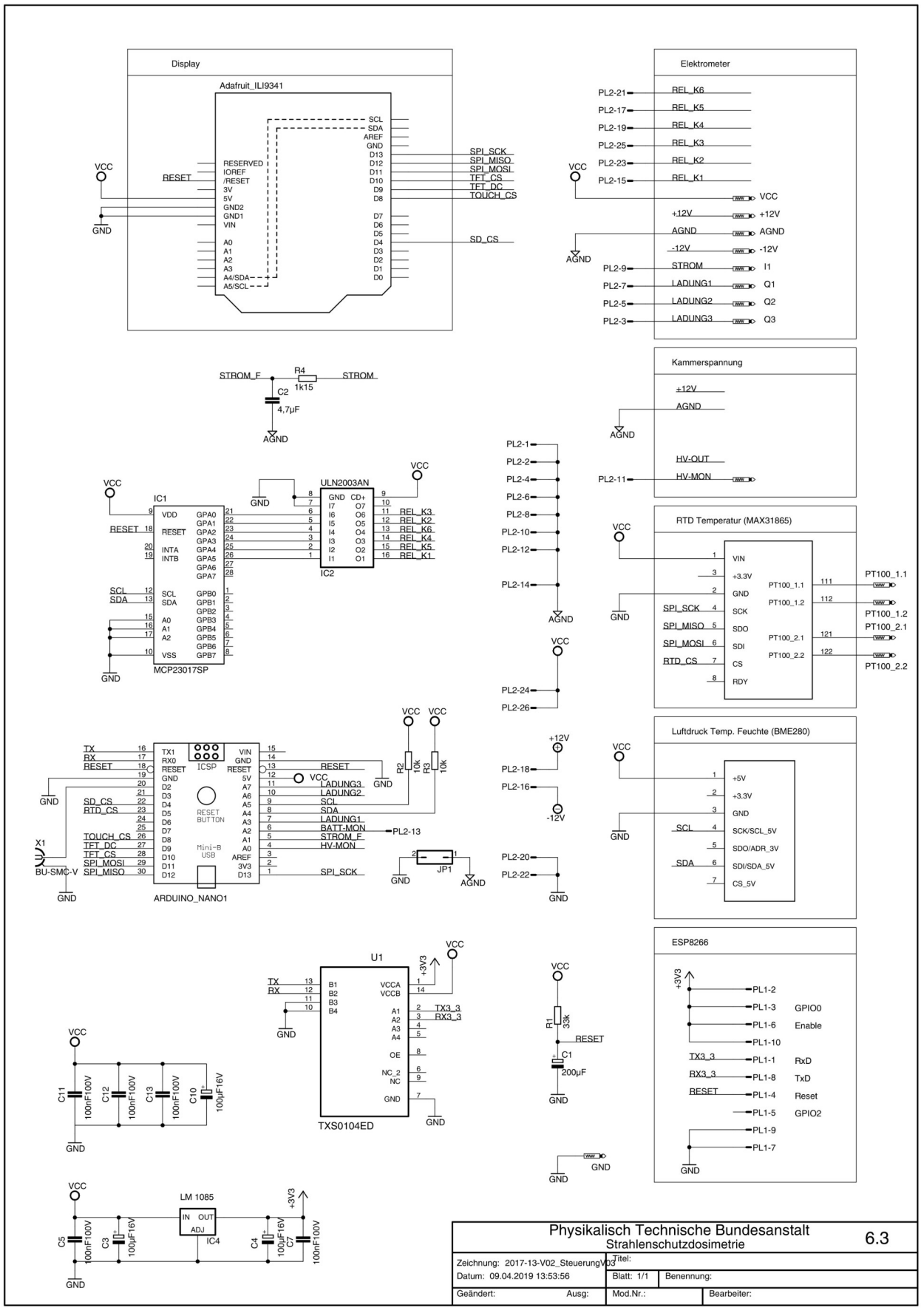
Platinenlayout



https://doi.org/10.7795/130.20210610

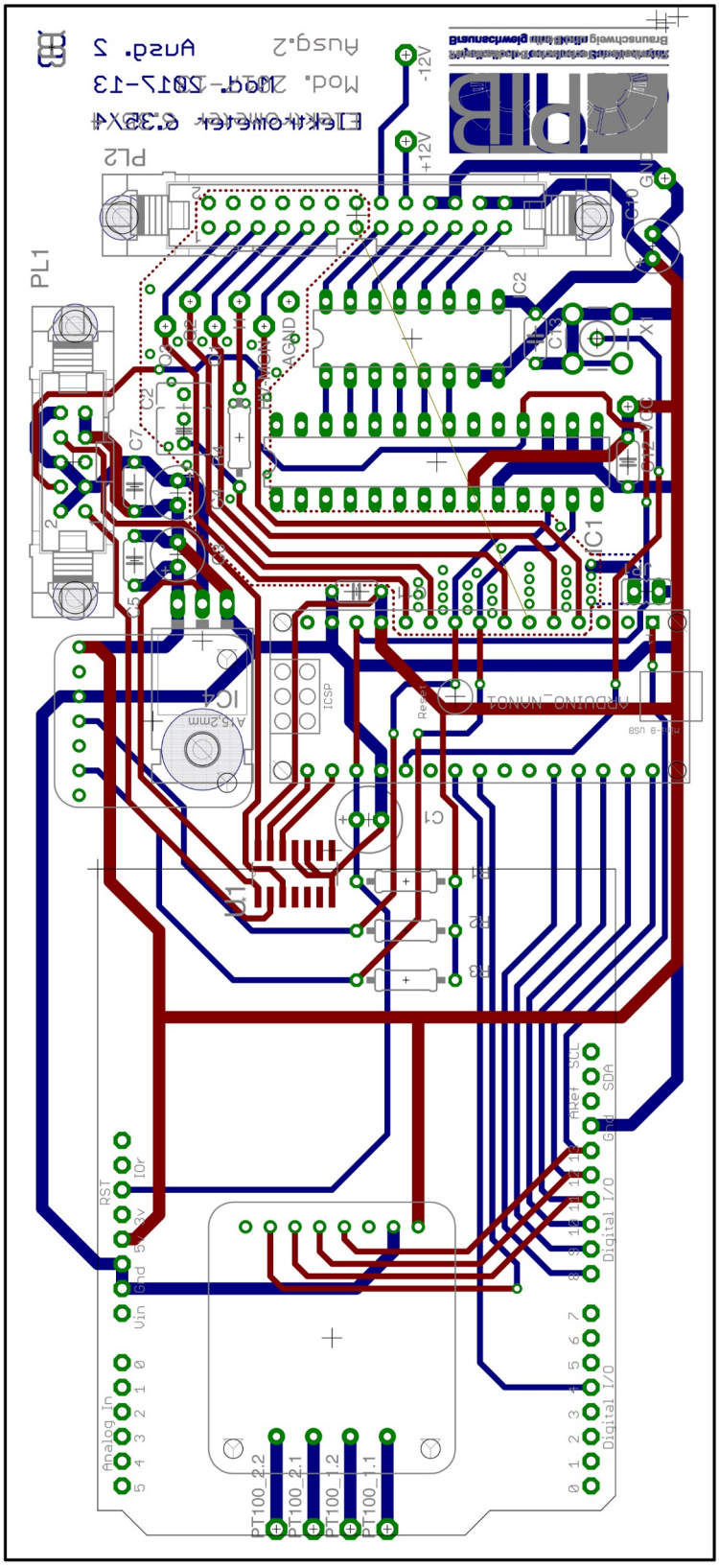
8.1.2 Steuerungsplatine

Schaltplan



https://doi.org/10.7795/130.20210610

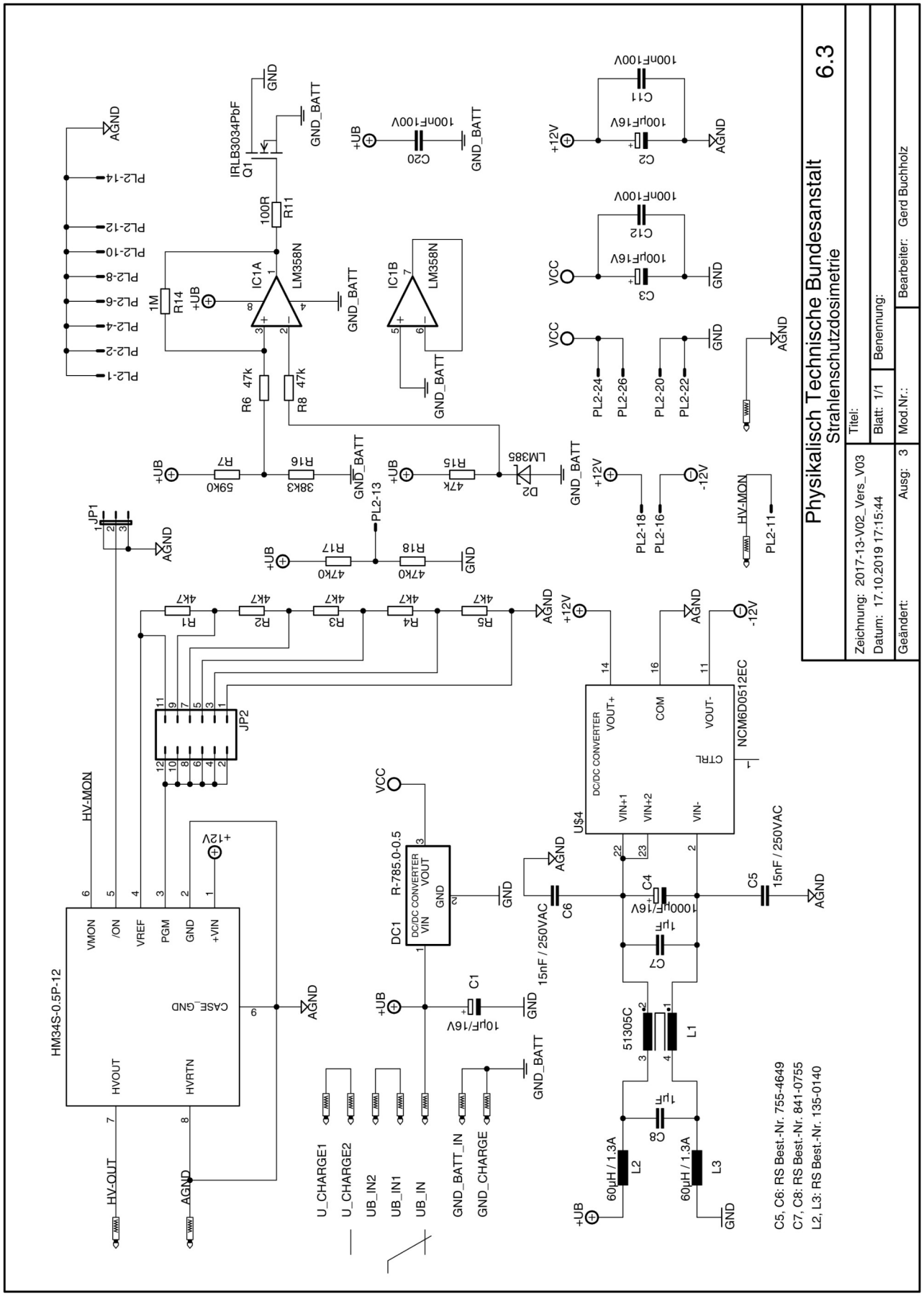
Platinenlayout



https://doi.org/10.7795/130.20210610

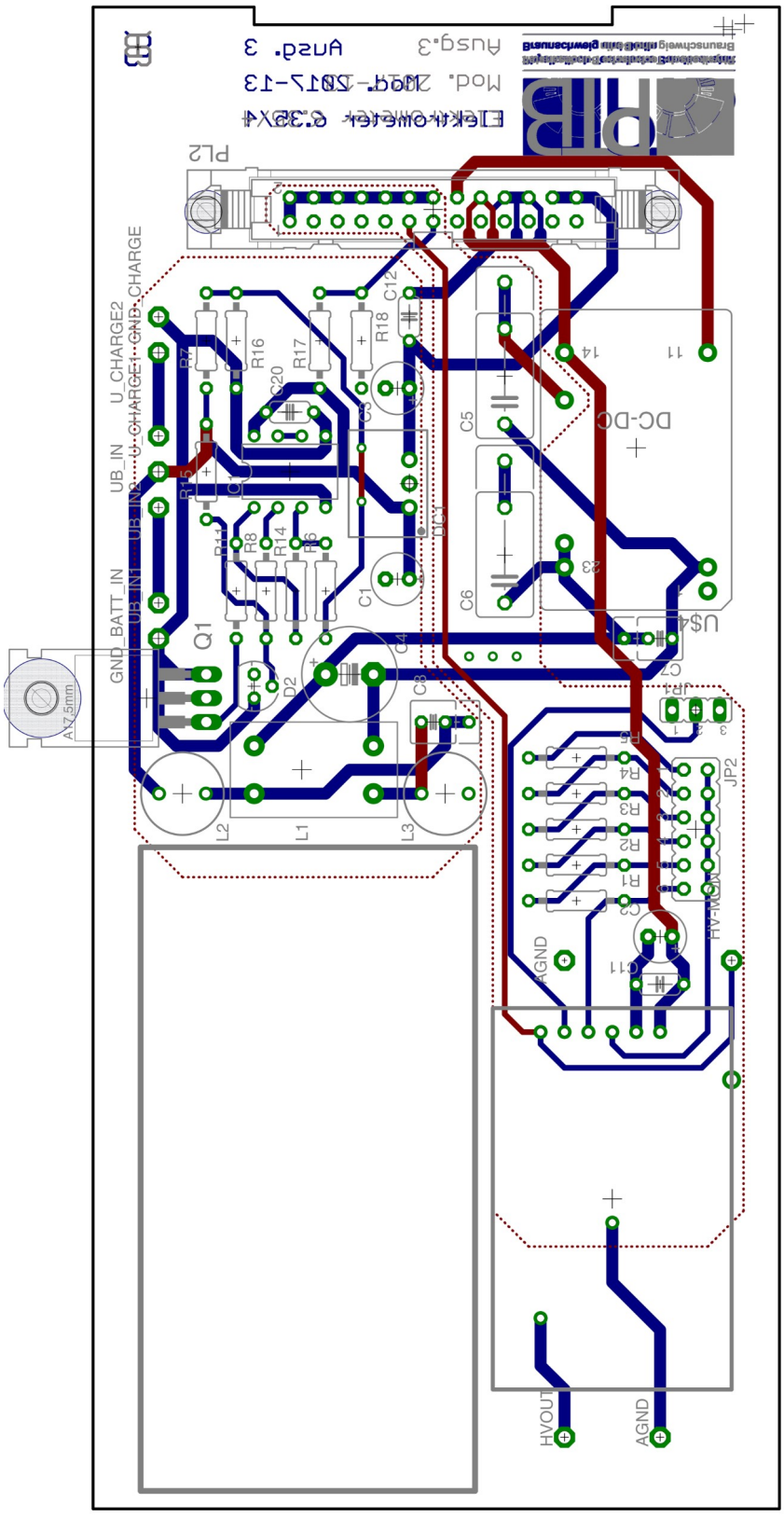
8.1.3 Versorgung

Schaltplan



https://doi.org/10.7795/130.20210610

Platinenlayout



https://doi.org/10.7795/130.20210610

8.2 Arduino Quelltexte

Es sind hier lediglich die speziell für diese Anwendung geschriebenen Quelltexte, aber nicht die der Standard-Bibliotheken und die von Adafruit für die Sensoren zur Verfügung gestellten Bibliotheken ausgedruckt.

8.2.1 Arduino Haupt-Programm

```
#include <SPI.h>
#include <Wire.h>
#include <stdint.h>
#include <Adafruit_GFX.h>
#include <Adafruit_ILI9341.h>
#include <Adafruit_STMPE610.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>
#include <Adafruit_MAX31865.h>
#include <Adafruit_MCP23017.h>

#include "Panel.h"
#include "Button.h"
#include "MillisTimer.h"
#include "StatFunct.h"

//#################################################################
// define: Strom- und Ladungs-Messung
//#################################################################

#define u_plus4V9 1000
#define u_plus0V1 18

#define MaxAusgSpgStrom 2.4
#define MaxAusgSpgLadung 2.4

#define KalFak_VV1000_VI100 1.246
#define KalFak_VV1000_VI10 1.117
#define KalFak_VV1000_VI1 1.153
#define KalFak_VV1_VI100 1.129
#define KalFak_VV1_VI10 1.093
#define KalFak_VV1_VI1 1.143

#define KalFak_VV1000_VQ100 0.857
#define KalFak_VV1000_VQ10 1.077
#define KalFak_VV1000_VQ1 1.021
#define KalFak_VV1_VQ100 0.978
#define KalFak_VV1_VQ10 1.133
#define KalFak_VV1_VQ1 1.027

#define KammerFaktor_mSv_C 3.430E+07 // in mSv/C

#define AnzMittelwertAnalog 15

#define IntervalAnalog 10  //  Zeit in ms fuer einen Analogwert aus function AnalogLesen etwa 6 ms
(AnalogLesen wird mit IntervalAnalog aufgerufen)
#define IntervalButton 100
#define IntervalPrint 1000 //  augegeben wird der Mittelwert aus AnzMittelwertAnalog Aufrufen von
AnalogLesen

#define UBattLow 6.1

#define HVMonKanal 0
#define StromKanal 1
#define BatterieKanal 2
#define Ladung1Kanal 3
#define Ladung2Kanal 6
#define Ladung3Kanal 7

#define REL_K1 5
#define REL_K2 1
#define REL_K3 0
#define REL_K4 3
#define REL_K5 4
#define REL_K6 2
```

https://doi.org/10.7795/130.20210610

```
//#################################################################
// define: Touch-Display
//#################################################################

// This is calibration data for the raw touch data to the screen coordinates
#define TS_MINX 150
#define TS_MINY 130
#define TS_MAXX 3800
#define TS_MAXY 4000

#define TFT_CS 10
#define TFT_DC 9

Adafruit_ILI9341 AdaTft = Adafruit_ILI9341(TFT_CS, TFT_DC);

#define STMPE_CS 8

Adafruit_STMPE610 AdaTouch = Adafruit_STMPE610(STMPE_CS);

/*
  // Color definitions
  #define ILI9341_BLACK       0x0000  ///<   0,   0,   0
  #define ILI9341_NAVY        0x000F  ///<   0,   0, 123
  #define ILI9341_DARKGREEN   0x03E0  ///<   0, 125,   0
  #define ILI9341_DARKCYAN    0x03EF  ///<   0, 125, 123
  #define ILI9341_MAROON      0x7800  ///< 123,   0,   0
  #define ILI9341_PURPLE      0x780F  ///< 123,   0, 123
  #define ILI9341_OLIVE       0x7BE0  ///< 123, 125,   0
  #define ILI9341_LIGHTGREY   0xC618  ///< 198, 195, 198
  #define ILI9341_DARKGREY    0x7BEF  ///< 123, 125, 123
  #define ILI9341_BLUE        0x001F  ///<   0,   0, 255
  #define ILI9341_GREEN       0x07E0  ///<   0, 255,   0
  #define ILI9341_CYAN        0x07FF  ///<   0, 255, 255
  #define ILI9341_RED         0xF800  ///< 255,   0,   0
  #define ILI9341_MAGENTA     0xF81F  ///< 255,   0, 255
  #define ILI9341_YELLOW      0xFFE0  ///< 255, 255,   0
  #define ILI9341_WHITE       0xFFFF  ///< 255, 255, 255
  #define ILI9341_ORANGE      0xFD20  ///< 255, 165,   0
  #define ILI9341_GREENYELLOW 0xAFE5  ///< 173, 255,  41
  #define ILI9341_PINK        0xFC18  ///< 255, 130, 198
*/

#define DisplayBackgroundColor ILI9341_BLUE
#define PanelBackgroundColor ILI9341_BLUE
#define PanelTextColor ILI9341_WHITE
#define PanelTextColorLowBat ILI9341_RED
#define PanelTextColorLowCurrent ILI9341_RED
#define ButtonColor ILI9341_WHITE
#define ButtonNotPressedColor ILI9341_LIGHTGREY
#define ButtonPressedColor ILI9341_DARKGREY
#define ButtonPressedTextColor ILI9341_WHITE
#define ButtonTextColor ILI9341_BLACK
#define ButtonResetNotPressedColor ILI9341_GREEN
#define ButtonResetPressedColor ILI9341_RED

// ButtonIndex
#define VorVerstX1 0
#define VorVerstX1000 1
#define StromX1 2
#define StromX10 3
#define StromX100 4
#define LadungX1 5
#define LadungX10 6
#define LadungX100 7
#define LadungReset 8

#define AnzButton 9

//#################################################################
// define: Luftdruck-, Temperatur- und Feuchte-Sensor
//#################################################################

#define RTD_CS 5

#define SEALEVELPRESSURE_HPA (1013.25)

#define LED_BUILTIN 3
```

https://doi.org/10.7795/130.20210610

```
//#################################################################
// var: Strom- und Ladungs-Messung
//#################################################################

TMillisTimer TimerReadAnalog;
TMillisTimer TimerCheckButton;
TMillisTimer TimerPrintValues;

float Temp;
float Feuchte;
float Luftdruck;
float TempPT100;
unsigned int jMittelwert;
unsigned int ULadung1[AnzMittelwertAnalog];
unsigned int ULadung2[AnzMittelwertAnalog];
unsigned int ULadung3[AnzMittelwertAnalog];
unsigned int UStrom[AnzMittelwertAnalog];
float UStromMW;
float ULadung1MW;
float ULadung2MW;
float ULadung3MW;
float StromMW;
float Ladung1MW;
float Ladung2MW;
float Ladung3MW;
int HVMon;
int Kammerspannung;
int BattMon;
float UBatt;
float frequency;
float ADCVerstaerkung;
float ADCOffset;
float FaktStrom;
float FaktLadung;
boolean VorVerst1000;

//#################################################################
// var: Dosis- und DosisLeistungs-Messung
//#################################################################

float Dosis_uSv;
float DosisLeistung_mSv_h;

//#################################################################
// var: Luftdruck-, Temperatur- und Feuchte-Messung
//#################################################################

Adafruit_BME280 bme; // I2C

Adafruit_MAX31865 PT100 = Adafruit_MAX31865(RTD_CS);  // use hardware SPI, just pass in the CS pin

// The value of the Rref resistor. Use 430.0 for PT100 and 4300.0 for PT1000
#define RREF      430.0
// The 'nominal' 0-degrees-C resistance of the sensor
// 100.0 for PT100, 1000.0 for PT1000
#define RNOMINAL  100.0


//#################################################################
// var: Digital I/O mit MCP23017
//#################################################################

Adafruit_MCP23017 mcp;


//#################################################################
// var: Frequenzausgabe
//#################################################################

#define pin 10
#define LowVoltage 0
#define HighVoltage 5
#define LowFrequency 1000
#define HighFrequency 30000
```

https://doi.org/10.7795/130.20210610

```
//#################################################################
// var: Touch-Display
//#################################################################

TPanel PanelStrom(10, 20, 220, 40, PanelBackgroundColor);  // Left, Top, Width, Hight
TPanel PanelLadung(10, 80, 220, 40, PanelBackgroundColor);
TPanel PanelLDruck(10, 140, 80, 20, PanelBackgroundColor);
TPanel PanelTemp(100, 140, 60, 20, PanelBackgroundColor);
TPanel PanelHVMon(170, 140, 60, 20, PanelBackgroundColor);

TPanel PanelEinheitDosisLeist(178, 220, 50, 40, PanelBackgroundColor);
TPanel PanelEinheitDosis(156, 270, 40, 40, PanelBackgroundColor);

TButton ButtonVorVerstX1(5, 170, 105, 40, ButtonColor);
TButton ButtonVorVerstX1000(130, 170, 105, 40, ButtonColor);
TButton ButtonStromX1(5, 220, 52, 40, ButtonColor);
TButton ButtonStromX10(63, 220, 52, 40, ButtonColor);
TButton ButtonStromX100(121, 220, 54, 40, ButtonColor);
TButton ButtonLadungX1(5, 270, 46, 40, ButtonColor);
TButton ButtonLadungX10(54, 270, 46, 40, ButtonColor);
TButton ButtonLadungX100(103, 270, 52, 40, ButtonColor);
TButton ButtonLadungReset(198, 270, 40, 40, ButtonColor);

TButton buttons[AnzButton];

unsigned int jButton;
int x;




//#################################################################
//
// setup
//
//#################################################################

void setup()
{
  x = 0;
  //---------------------------------------------------------------
  // Serial initialisieren
  //---------------------------------------------------------------

  Serial.begin(115200);

  //---------------------------------------------------------------
  // TFT
  //---------------------------------------------------------------

  //  Serial.println("InitScreen");

  AdaTft.begin();
  AdaTft.setRotation(2);  // Touch-Koordinaten muessen auch gedreht werden; siehe CheckTouch
  AdaTft.fillScreen(DisplayBackgroundColor);

  if (!AdaTouch.begin()) {
    Serial.println("Unable to start touchscreen.");
  }
  else {
    //    Serial.println("Touchscreen started.");
  }


  //---------------------------------------------------------------
  // Luftdruck Temperatur Feuchte BME280
  //---------------------------------------------------------------
  // default settings
  // (you can also pass in a Wire library object like &Wire2)
  bool status;
  status = bme.begin();
  if (!status) {
    AdaTft.setCursor(3, 3);
    AdaTft.setTextColor(ILI9341_WHITE);
    AdaTft.setTextSize(2);
    AdaTft.println("Could not find a valid BME280 sensor, check wiring!");
    while (1);
  }
```

https://doi.org/10.7795/130.20210610

```
//-------------------------------------------------------------
// Temperatur PT100 mit MAX31865
//-------------------------------------------------------------
PT100.begin(MAX31865_4WIRE);  // set to 2WIRE or 4WIRE as necessary

//-------------------------------------------------------------
// MCP23017
//-------------------------------------------------------------
mcp.begin();      // use default address 0

mcp.pinMode(0, OUTPUT);
mcp.pinMode(1, OUTPUT);
mcp.pinMode(2, OUTPUT);
mcp.pinMode(3, OUTPUT);
mcp.pinMode(4, OUTPUT);
mcp.pinMode(5, OUTPUT);
mcp.pinMode(6, INPUT);
mcp.pullUp(6, HIGH);
mcp.pinMode(7, INPUT);
mcp.pullUp(7, HIGH);

mcp.pinMode(8, INPUT);
mcp.pullUp(8, HIGH);
mcp.pinMode(9, INPUT);
mcp.pullUp(9, HIGH);
mcp.pinMode(10, INPUT);
mcp.pullUp(10, HIGH);
mcp.pinMode(11, INPUT);
mcp.pullUp(11, HIGH);
mcp.pinMode(12, INPUT);
mcp.pullUp(12, HIGH);
mcp.pinMode(13, INPUT);
mcp.pullUp(13, HIGH);
mcp.pinMode(14, INPUT);
mcp.pullUp(14, HIGH);
mcp.pinMode(15, INPUT);
mcp.pullUp(15, HIGH);

mcp.digitalWrite(REL_K1, HIGH);
mcp.digitalWrite(REL_K2, LOW);
mcp.digitalWrite(REL_K3, LOW);
mcp.digitalWrite(REL_K4, HIGH);
mcp.digitalWrite(REL_K5, LOW);
mcp.digitalWrite(REL_K6, HIGH);

//-------------------------------------------------------------
// TFT Benutzeroberflaeche initialisieren
//-------------------------------------------------------------
PanelStrom.setRectangle();
PanelLadung.setRectangle();
PanelLDruck.setRectangle();
PanelTemp.setRectangle();
PanelHVMon.setRectangle();

PanelStrom.BackgroundColor(DisplayBackgroundColor);
PanelLadung.BackgroundColor(DisplayBackgroundColor);
PanelLDruck.BackgroundColor(DisplayBackgroundColor);
PanelTemp.BackgroundColor(DisplayBackgroundColor);
PanelHVMon.BackgroundColor(DisplayBackgroundColor);
PanelEinheitDosisLeist.BackgroundColor(DisplayBackgroundColor);
PanelEinheitDosis.BackgroundColor(DisplayBackgroundColor);


PanelStrom.TextHight(3);
PanelLadung.TextHight(3);
PanelLDruck.TextHight(1);
PanelTemp.TextHight(1);
PanelHVMon.TextHight(1);
PanelEinheitDosisLeist.TextHight(2);
PanelEinheitDosis.TextHight(2);

PanelStrom.TextColor(PanelTextColor);
PanelLadung.TextColor(PanelTextColor);
PanelLDruck.TextColor(PanelTextColor);
PanelTemp.TextColor(PanelTextColor);
PanelHVMon.TextColor(PanelTextColor);
PanelEinheitDosisLeist.TextColor(PanelTextColor);
PanelEinheitDosis.TextColor(PanelTextColor);
```

https://doi.org/10.7795/130.20210610

```
  buttons[VorVerstX1] = ButtonVorVerstX1;
  buttons[VorVerstX1000] = ButtonVorVerstX1000;
  buttons[StromX1] = ButtonStromX1;
  buttons[StromX10] = ButtonStromX10;
  buttons[StromX100] = ButtonStromX100;
  buttons[LadungX1] = ButtonLadungX1;
  buttons[LadungX10] = ButtonLadungX10;
  buttons[LadungX100] = ButtonLadungX100;
  buttons[LadungReset] = ButtonLadungReset;

  for (jButton = 0; jButton <= AnzButton - 1; jButton++)
  {
    buttons[jButton].setRectangle();
    buttons[jButton].TextHight(2);
    buttons[jButton].TextColor(ButtonTextColor);
    buttons[jButton].BackgroundColor(ButtonNotPressedColor);
  }
  buttons[LadungReset].TextColor(ButtonPressedTextColor);


  //-------------------------------------------------------------
  // Messbereiche initialisieren
  //-------------------------------------------------------------
  Bereich(VorVerstX1);
  Bereich(StromX1);
  Bereich(LadungX1);
  Bereich(LadungReset);


  //-------------------------------------------------------------
  // Timer initialisieren
  //-------------------------------------------------------------
  TimerReadAnalog.Interval(IntervalAnalog);
  TimerReadAnalog.Start();

  TimerCheckButton.Interval(IntervalButton);
  TimerCheckButton.Start();

  TimerPrintValues.Interval(IntervalPrint);
  TimerPrintValues.Start();


  //-------------------------------------------------------------
  // Daten initialisieren
  //-------------------------------------------------------------
  for (jMittelwert = 0; jMittelwert < AnzMittelwertAnalog - 1; jMittelwert++)
  {
    ULadung1[jMittelwert] = 0;
    UStrom[jMittelwert] = 0;
  }
  jMittelwert = 0;


  //-------------------------------------------------------------
  // ADC initialisieren
  //-------------------------------------------------------------
  ADCVerstaerkung =  4.8 / (u_plus4V9 - u_plus0V1);
  ADCOffset = 0.1 - ADCVerstaerkung * u_plus0V1;


  //-------------------------------------------------------------
  // LED initialisieren
  //-------------------------------------------------------------
  pinMode(LED_BUILTIN, OUTPUT);

  //-------------------------------------------------------------
  // Tabellenkopf
  //-------------------------------------------------------------
  Serial.print("t in s ;  ");
  Serial.print("H/t in mSv/h ;  "),
               Serial.print("H in µSv ;  ");
  Serial.print("I in pA ;  "),
               Serial.print("Q in pC ;  ");
  Serial.print("P in hPa ;  ");
  Serial.print("T in Cel ;  ");
  Serial.print("U_Kammer in V ;  ");
  Serial.println("U_Batt in V ;  ");

}
```

https://doi.org/10.7795/130.20210610

```
//###############################################################
//
// loop
//
//###############################################################

void loop()
{
  //-------------------------------------------------------------
  // Timer Analog
  //-------------------------------------------------------------
  if (TimerReadAnalog.AbgelaufenEinzel() == true)
  {
    UStrom[jMittelwert] = AnalogLesen(StromKanal);
    ULadung1[jMittelwert] = AnalogLesen(Ladung1Kanal);

    //    // Frequenz
    //
    //    frequency = map(Strom[jMittelwert], LowVoltage, HighVoltage, LowFrequency, HighFrequency);
    //    tone(pin, frequency);

    jMittelwert++;
    if (jMittelwert > (AnzMittelwertAnalog - 1))
    {
      jMittelwert = 0;
    }
    TimerReadAnalog.Start();

    digitalWrite(LED_BUILTIN, HIGH);
  }

  //-------------------------------------------------------------
  // Timer Button
  //-------------------------------------------------------------
  if (TimerCheckButton.AbgelaufenEinzel() == true)
  {
    TimerCheckButton.Start();
    CheckTouch();
  }

  //-------------------------------------------------------------
  // Timer Ausgabe
  //-------------------------------------------------------------
  if (TimerPrintValues.AbgelaufenEinzel() == true)
  {
    TimerPrintValues.Start();
    //    Temp = bme.readTemperature();
    //    Feuchte = bme.readHumidity();
    Luftdruck = bme.readPressure() / 100.0;

    TempPT100 = PT100.temperature(RNOMINAL, RREF);
    HVMon = AnalogLesen(HVMonKanal);
    Kammerspannung = int((AnalogInSpannung(float(HVMon))) * 100);

    BattMon = AnalogLesen(BatterieKanal);
    UBatt = AnalogInSpannung(float(BattMon)) * 2;  //  Batteriespannung wird mit Teiler 2 zu 1
gemessen

    printValues();
    digitalWrite(LED_BUILTIN, LOW);
  }
}
```

https://doi.org/10.7795/130.20210610

```
//#############################################################
//
// Functions
//
//#############################################################


int AnalogLesen(int AnalogPort)
{
  int jAnalog;
  unsigned int AnalogWerte[5];
  unsigned int MedianWert;

  AnalogWerte[0] = analogRead(AnalogPort);
  delay(1);
  for (jAnalog = 0; jAnalog < 5; jAnalog++)
  {
    AnalogWerte[jAnalog] = analogRead(AnalogPort);
    delay(1);
  }
  return Median(AnalogWerte, 5);
}



float AnalogInSpannung(float AnalogWert)
{
  float Spannung;

  Spannung = AnalogWert * ADCVerstaerkung + ADCOffset;

  return Spannung;
}



void printValues()
{
  UStromMW = Mittelwert(UStrom, AnzMittelwertAnalog);
  UStromMW = AnalogInSpannung(UStromMW);
  if (UStromMW > MaxAusgSpgStrom)
  {
    StromMW = 999.9e-9;
    DosisLeistung_mSv_h = 999.9;
  }
  else
  {
    StromMW = UStromMW * FaktStrom;
    DosisLeistung_mSv_h = StromMW * KammerFaktor_mSv_C * 3600 * 1013.25 / Luftdruck * (TempPT100 +
273.15) / 293.15;
  }

  ULadung1MW = Mittelwert(ULadung1, AnzMittelwertAnalog);
  ULadung1MW = AnalogInSpannung(ULadung1MW);
  if (ULadung1MW > MaxAusgSpgLadung)
  {
    Ladung1MW = 999.9e-9;
    Dosis_uSv = 999.9;
  }
  else
  {
    Ladung1MW = ULadung1MW * FaktLadung;
    Dosis_uSv = Ladung1MW * KammerFaktor_mSv_C * 1000 * 1013.25 / Luftdruck *
    (TempPT100 + 273.15) / 293.15;
  }

  if (UStromMW < 0.2)
  {
    PanelStrom.TextColor(PanelTextColorLowCurrent);
  }
  else
  {
    PanelStrom.TextColor(PanelTextColor);
  }

  if (ULadung1MW < 0.2)
  {
    PanelLadung.TextColor(PanelTextColorLowCurrent);
  }
```

https://doi.org/10.7795/130.20210610

```
  else
  {
    PanelLadung.TextColor(PanelTextColor);
  }

  if (UBatt < UBattLow)
  {
    PanelLDruck.TextColor(PanelTextColorLowBat);
    PanelTemp.TextColor(PanelTextColorLowBat);
    PanelHVMon.TextColor(PanelTextColorLowBat);
  }

  PanelStrom.Text(Format(DosisLeistung_mSv_h) + " mSv/h");
  PanelLadung.Text(Format(Dosis_uSv) + " uSv");

  PanelLDruck.Text(String(Luftdruck) + " hPa");
  PanelTemp.Text(String(TempPT100) + " Cel");
  PanelHVMon.Text(String(Kammerspannung) + " V");

  Serial.print(millis() / 1000);
  Serial.print(" ;   ");
  Serial.print(String(DosisLeistung_mSv_h, 4));
  Serial.print(" ;   ");
  Serial.print(String(Dosis_uSv, 4));
  Serial.print(" ;   ");
  Serial.print(String(StromMW * 1E12, 3));
  Serial.print(" ;   ");
  Serial.print(String(Ladung1MW * 1E12, 3));
  Serial.print(" ;   ");
  Serial.print(Luftdruck);
  Serial.print(" ;   ");
  Serial.print(TempPT100);
  Serial.print(" ;   ");
  Serial.print(Kammerspannung);
  Serial.print(" ;   ");
  Serial.print(UBatt);
  Serial.println(" ;   ");
}


String Format(float Wert)
{
  String Ausgabe;
  String Prefix;

  if (Wert >= 100)
  {
    Ausgabe = (String(Wert, 1) + Prefix);
  }
  else if (Wert >= 10)
  {
    Ausgabe = (String(Wert, 2) + Prefix);
  }
  else if (Wert >= 1)
  {
    Ausgabe = (String(Wert, 3) + Prefix);
  }
  else
  {
    Ausgabe = (String(Wert, 4) + Prefix);
  }
  return Ausgabe;
}


void CheckTouch()
{
  byte jButton = 0;
  byte ActiveButton;

  //  [0] ButtonVorVerstX1
  //  [1] ButtonVorVerstX1000
  //  [2] ButtonStromX1
  //  [3] ButtonStromX10
  //  [4] ButtonStromX100
  //  [5] ButtonLadungX1
  //  [6] ButtonLadungX10
  //  [7] ButtonLadungX100
  //  [8] ButtonLadungReset
```

https://doi.org/10.7795/130.20210610

```
  if (!AdaTouch.bufferEmpty())
  {
    TS_Point p = AdaTouch.getPoint();
    p.x = map(p.x, TS_MINX, TS_MAXX, 0, 240);
    p.y = map(p.y, TS_MINY, TS_MAXY, 0, 320);

    p.x = 240 - p.x;  // Umrechnung der Touch-Koordinaten wegen Drehung des Displays;
                         siehe TFT im setup
    p.y = 320 - p.y;

    jButton = 0;
    ActiveButton = 255;
    do
    {
      if (buttons[jButton].buttonPressed(p.x, p.y))
      {
        ActiveButton = jButton;
      }
      jButton++;
    }
    while ((jButton < AnzButton) && (ActiveButton >= 255));
    if (ActiveButton < AnzButton)
    {
      Bereich(ActiveButton);
    }
  }
}


void Bereich(byte Btn)
{
  if (Btn == 2 || Btn == 3 || Btn == 4)
  {
    for (jButton = 2; jButton <= 4; jButton++)
    {
      buttons[jButton].BackgroundColor(ButtonNotPressedColor);
      buttons[jButton].TextColor(ButtonTextColor);
    }
  }

  if (Btn == 5 || Btn == 6 || Btn == 7)
  {
    for (jButton = 5; jButton <= 7; jButton++)
    {
      buttons[jButton].BackgroundColor(ButtonNotPressedColor);
      buttons[jButton].TextColor(ButtonTextColor);
    }
    buttons[LadungReset].BackgroundColor(ButtonResetNotPressedColor);
  }

  switch (Btn)
  {
    case 0:
      buttons[VorVerstX1].BackgroundColor(ButtonPressedColor);
      buttons[VorVerstX1].TextColor(ButtonPressedTextColor);
      buttons[VorVerstX1000].BackgroundColor(ButtonNotPressedColor);
      buttons[VorVerstX1000].TextColor(ButtonTextColor);
      mcp.digitalWrite(REL_K1, HIGH);
      VorVerst1000 = false;
      Bereich(LadungReset);

      for (jButton = 2; jButton <= 7; jButton++)
      {
        buttons[jButton].BackgroundColor(ButtonNotPressedColor);
        buttons[jButton].TextColor(ButtonTextColor);
      }

      FaktStrom = 0;
      FaktLadung = 0;
      break;

    case 1:
      buttons[VorVerstX1000].BackgroundColor(ButtonPressedColor);
      buttons[VorVerstX1000].TextColor(ButtonPressedTextColor);
      buttons[VorVerstX1].BackgroundColor(ButtonNotPressedColor);
      buttons[VorVerstX1].TextColor(ButtonTextColor);
      mcp.digitalWrite(REL_K1, LOW);
      VorVerst1000 = true;
```

https://doi.org/10.7795/130.20210610

```
  Bereich(LadungReset);
  for (jButton = 2; jButton <= 7; jButton++)
  {
    buttons[jButton].BackgroundColor(ButtonNotPressedColor);
    buttons[jButton].TextColor(ButtonTextColor);
  }
  FaktStrom = 0;
  FaktLadung = 0;
  break;

case 2:
  mcp.digitalWrite(REL_K2, HIGH);
  mcp.digitalWrite(REL_K3, HIGH);
  buttons[StromX1].BackgroundColor(ButtonPressedColor);
  buttons[StromX1].TextColor(ButtonPressedTextColor);
  if (VorVerst1000 == true) {
    FaktStrom = KalFak_VV1000_VI1 * 1.0E-11;
  }  //   * 1.0E-11 * 1.0E-00
  else {
    FaktStrom = KalFak_VV1_VI1 * 1.0E-08;
  }     //   * 1.0E-08 * 1.0E-00
  break;

case 3:
  mcp.digitalWrite(REL_K2, HIGH);
  mcp.digitalWrite(REL_K3, LOW);
  buttons[StromX10].BackgroundColor(ButtonPressedColor);
  buttons[StromX10].TextColor(ButtonPressedTextColor);
  if (VorVerst1000 == true) {
    FaktStrom = KalFak_VV1000_VI10 * 1.0E-12;
  }  //   * 1.0E-11 * 1.0E-01
  else {
    FaktStrom = KalFak_VV1_VI10 * 1.0E-9;
  }      //   * 1.0E-11 * 1.0E-01
  break;

case 4:
  mcp.digitalWrite(REL_K2, LOW);
  mcp.digitalWrite(REL_K3, LOW);
  buttons[StromX100].BackgroundColor(ButtonPressedColor);
  buttons[StromX100].TextColor(ButtonPressedTextColor);
  if (VorVerst1000 == true) {
    FaktStrom = KalFak_VV1000_VI100 * 1.0E-13;
  }  //   * 1.0E-11 * 1.0E-02
  else {
    FaktStrom = KalFak_VV1_VI100 * 1.0E-10;
  }     //   * 1.0E-11 * 1.0E-02
  break;

case 5:
  mcp.digitalWrite(REL_K4, HIGH);
  mcp.digitalWrite(REL_K5, LOW);
  mcp.digitalWrite(REL_K6, LOW);
  buttons[LadungX1].BackgroundColor(ButtonPressedColor);
  buttons[LadungX1].TextColor(ButtonPressedTextColor);
  if (VorVerst1000 == true) {
    FaktLadung = KalFak_VV1000_VQ1 * 1.0E-10;
  }  //   * 1.0E-11 * 1.0E+07 * 1E-06
  else {
    FaktLadung = KalFak_VV1_VQ1 * 1.0E-07;
  }     //   * 1.0E-08 * 1.0E+07 * 1E-06
  break;

case 6:
  mcp.digitalWrite(REL_K4, LOW);
  mcp.digitalWrite(REL_K5, HIGH);
  mcp.digitalWrite(REL_K6, LOW);
  buttons[LadungX10].BackgroundColor(ButtonPressedColor);
  buttons[LadungX10].TextColor(ButtonPressedTextColor);
  if (VorVerst1000 == true) {
    FaktLadung = KalFak_VV1000_VQ10 * 1.0E-11;
  }  //   * 1.0E-11 * 1.0E+07 * 1E-07
  else {
    FaktLadung = KalFak_VV1_VQ10 * 1.0E-08;
  }     //   * 1.0E-08 * 1.0E+07 * 1E-07
  break;

case 7:
  mcp.digitalWrite(REL_K4, LOW);
```

https://doi.org/10.7795/130.20210610

```
      mcp.digitalWrite(REL_K5, LOW);
      mcp.digitalWrite(REL_K6, LOW);
      buttons[LadungX100].BackgroundColor(ButtonPressedColor);
      buttons[LadungX100].TextColor(ButtonPressedTextColor);
      if (VorVerst1000 == true) {
        FaktLadung = KalFak_VV1000_VQ100 * 1.0E-12;
      }  //   * 1.0E-11 * 1.0E+07 * 1E-08
      else {
        FaktLadung = KalFak_VV1_VQ100 * 1.0E-09;
      }     //   * 1.0E-08 * 1.0E+07 * 1E-08
      break;

    case 8:
      mcp.digitalWrite(REL_K6, HIGH);
      buttons[LadungReset].BackgroundColor(ButtonResetPressedColor);
      break;
  }

  buttons[VorVerstX1].Text("VorV1");
  buttons[VorVerstX1000].Text("VorV1000");

  PanelEinheitDosisLeist.Text("mSv/h");
  PanelEinheitDosis.Text("uSv");

  if (VorVerst1000 == true) {
    buttons[StromX1].Text("3");
    buttons[StromX10].Text("0,3");
    buttons[StromX100].Text("0,03");
    buttons[LadungX1].Text("8");
    buttons[LadungX10].Text("0,8");
    buttons[LadungX100].Text("0,08");
    buttons[LadungReset].Text("Rst");
  }
  else {
    buttons[StromX1].Text("3 k");
    buttons[StromX10].Text("300");
    buttons[StromX100].Text("30");
    buttons[LadungX1].Text("8 k");
    buttons[LadungX10].Text("800");
    buttons[LadungX100].Text("80");
    buttons[LadungReset].Text("Rst");
  }
}
```

https://doi.org/10.7795/130.20210610

8.2.2 Panel.h

```
#ifndef __Panel__
#define __Panel__

#include <Arduino.h>
#include <stdint.h>
#include <Adafruit_ILI9341.h>
#include <Adafruit_STMPE610.h>

extern Adafruit_ILI9341 AdaTft;

class TPanel
{
  protected:
    unsigned int Left;
    unsigned int Top;
    unsigned int Width;
    unsigned int Hight;
    unsigned int Color;
    unsigned int ColorBackground;
    unsigned int ColorText;
    unsigned int HightText;

  public:
    TPanel ()
    {
      Left = 0;
      Top = 0;
      Width = 33;
      Hight = 33;
      Color = ILI9341_WHITE;
      ColorBackground = ILI9341_LIGHTGREY;
      ColorText = ILI9341_WHITE;
      HightText = 3;
    }

    TPanel (unsigned int Left, unsigned int Top, unsigned int Width, unsigned int Hight,
            unsigned int Color);

    void setRectangle();
    void Text(String TextString);
    void RectColor(unsigned int ColorR);
    void BackgroundColor(unsigned int ColorB);
    void TextColor(unsigned int ColorT);
    void TextHight(unsigned int HightT);
};

#endif
```

https://doi.org/10.7795/130.20210610

8.2.3 Panel.cpp

```
#include "Panel.h"

TPanel::TPanel(unsigned int Left, unsigned int Top, unsigned int Width, unsigned int Hight, unsigned
int Color)
{
  this->Left  = Left;
  this->Top   = Top;
  this->Width = Width;
  this->Hight = Hight;
  this->Color = Color;
}



void TPanel::setRectangle()
{
  AdaTft.drawRect(Left, Top, Width, Hight, Color);
}



void TPanel::Text(String TextString)
{
  char TFTCharArray[35];
  unsigned int TextTop;
  unsigned int TextLeft;
  unsigned int TextLength;
  int TextXSpace;
  int TextYSpace;
  String Dummy;

  TextLength = TextString.length();

  TextYSpace = HightText * 8;
  if (TextYSpace <= Hight)
  {
    TextTop = Top + ((Hight - TextYSpace) / 2);
  }
  else
  {
    TextTop = Top;
  }

  TextXSpace = TextLength * HightText * 6;
  if (TextXSpace <= Width)
  {
    TextLeft = Left + ((Width - TextXSpace) / 2);
  }
  else
  {
    TextLeft = Left;
  }

  AdaTft.fillRect(Left, TextTop, Width, TextYSpace, ColorBackground);
  AdaTft.drawRect(Left, Top, Width, Hight, Color);
  AdaTft.setTextColor(ColorText);
  AdaTft.setTextSize(HightText);
  AdaTft.setCursor(TextLeft , TextTop);
  AdaTft.println(TextString);
}



void TPanel::RectColor(unsigned int ColorR)
{
  Color = ColorR;
}

void TPanel::BackgroundColor(unsigned int ColorB)
{
  ColorBackground = ColorB;
  AdaTft.fillRect(Left, Top, Width, Hight, ColorBackground);
  AdaTft.drawRect(Left, Top, Width, Hight, Color);
}
```

https://doi.org/10.7795/130.20210610

```
void TPanel::TextColor(unsigned int ColorT)
{
  ColorText = ColorT;
}



void TPanel::TextHight(unsigned int HightT)
{
  HightText = HightT;
}
```

https://doi.org/10.7795/130.20210610

8.2.4 Button.h

```
#ifndef __Button__
#define __Button__

#include <Arduino.h>
#include <stdint.h>
#include <Adafruit_ILI9341.h>
#include <Adafruit_STMPE610.h>
#include "Panel.h"

class TButton : public TPanel
{
  public:

    TButton() : TPanel()
    {
      Left = 0;
      Top = 0;
      Width = 0;
      Hight = 0;
      Color = 0;
      ColorText = 0;
      HightText = 0;
    }

    TButton (unsigned int Left, unsigned int Top, unsigned int Width, unsigned int Hight, unsigned
int Color) : TPanel(Left, Top, Width, Hight, Color)
    {
      ColorText = ILI9341_BLACK;
      HightText = 3;
    }

    bool buttonPressed(unsigned int px, unsigned int py);

};

#endif
```

8.2.5 Button.cpp

```
#include "Button.h"

bool TButton::buttonPressed(unsigned int px, unsigned int py)
{
  if (  Left < px && (Left + Width) > px &&
        Top < py && (Top + Hight) > py)
      {
        return true;
      }
  return false;
}
```

https://doi.org/10.7795/130.20210610

8.2.6 MillisTimer.h

```
#ifndef __MillisTimer__
#define __MillisTimer__

#include <Arduino.h>
#include <stdint.h>

class TMillisTimer
{
  private:
    unsigned int MillisStart;
    unsigned int MillisInterval;

  public:
    TMillisTimer ();

    void Interval(unsigned int Time_ms);
    void Start();
    boolean AbgelaufenNeustart();
    boolean AbgelaufenEinzel();
};

#endif
```

8.2.7 MillisTimer.cpp

```
#include "MillisTimer.h"

TMillisTimer::TMillisTimer ()
{
}

void TMillisTimer::Interval(unsigned int Time_ms)
{
  MillisInterval = Time_ms;
}

void TMillisTimer::Start()
{
  MillisStart = millis();
}

boolean TMillisTimer::AbgelaufenNeustart()
{
  boolean result;
  unsigned int MillisAktuell;

  result = false;
  MillisAktuell = millis();
  if ((MillisAktuell - MillisStart) >= MillisInterval)
  {
    result = true;
    MillisStart = MillisStart + MillisInterval;
  }
  return result;
}

boolean TMillisTimer::AbgelaufenEinzel()
{
  boolean result;
  unsigned int MillisAktuell;

  result = false;
  MillisAktuell = millis();
  if ((MillisAktuell - MillisStart) >= MillisInterval)
  {
    result = true;
  }
  return result;
}
```

https://doi.org/10.7795/130.20210610

8.2.8 StatFunct.h

```
#ifndef __Median__
#define __Median__

#include <Arduino.h>
#include <stdint.h>

unsigned int Median(unsigned int Werte[], int AnzWerte);
unsigned int Sort(unsigned int Werte[], int AnzWerte);
float Mittelwert(unsigned int Werte[], int AnzWerte);

#endif
```

8.2.9 StatFunct.cpp

```
#ifndef __Median__
#define __Median__

#include <Arduino.h>
#include <stdint.h>

int jSort0;
int jSort1;
float Summe;

unsigned int Median(unsigned int Werte[], int AnzWerte)
{
  unsigned int dummy;
  boolean getauscht;

  for (jSort0 = 0; jSort0 < AnzWerte; jSort0++)
  {
    //        Serial.print("In ");
    //        Serial.println(Werte[jSort0]);
  }

  getauscht = true;
  jSort1 = 0;
  do
  {
    getauscht = false;
    for (jSort0 = 0; jSort0 < AnzWerte - 1; jSort0++)
    {
      if (Werte[jSort0] > Werte[jSort0 + 1])
      {
        dummy = Werte[jSort0 + 1];
        Werte[jSort0 + 1] = Werte[jSort0];
        Werte[jSort0] = dummy;
        getauscht = true;
      }
    }
    jSort1++;
  }  while ((jSort1 < AnzWerte) && (getauscht == true));

  //  Serial.print("getauscht ");
  //  Serial.println(jSort1);
  for (jSort0 = 0; jSort0 < AnzWerte; jSort0++)
  {
    //    Serial.println(Werte[jSort0]);
  }

  return Werte[AnzWerte / 2];
}

float Mittelwert(unsigned int Werte[], int AnzWerte)
{
  Summe = 0;
  for (jSort0 = 0; jSort0 < AnzWerte - 1; jSort0++)
  {
    Summe = Summe + Werte[jSort0];
  }
  return Summe / AnzWerte;
}

#endif
```

https://doi.org/10.7795/130.20210610

Herausgeber und Verlag:
Physikalisch-Technische Bundesanstalt
ISNI: 0000 0001 2186 1887

Bundesallee 100
38116 Braunschweig

Fachbereich 6.3
Strahlenschutzdosimetrie
Telefon: 05 31 592-63 01
Telefax: 05 31 592-60 15
www.ptb.de