HOCHSCHULE
ANHALT University
of Applied Sciences

FB6 - EMW
Fachbereich Elektrotechnik, Maschinenbau und Wirtschaftsingenieurwesen
Campus Köthen

Belegarbeit

Für den Abschluss des Moduls
Programmierung Eingebetter Systeme

Adrian Frank Böschel

Vorname Nachname

Elektro- und Informationstechnik, EIT21, 5058600

Studiengang, Matrikel, Matrikelnummer

Thema:

Digitales Stethoskop mit einem RaspberryPi Pico

Prof. Dr. Chmielewski

Hochschulmentor(in)

23.09.2024

Abgabe am

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

HOCHSCHULE ANHALT University of Applied Sciences

Name, Vorname: Böschel, Adrian Frank
Matrikelnummer: 5058600
Studiengang: Elektro- und Informationstechnik

Selbstständigkeitserklärung

Durch meine Unterschrift erkläre ich, dass ich die vorliegende Arbeit mit dem Titel
Erstellen eines Digitalen Stethoskop mit einem Raspberry Pi Pico

selbständig verfasst und in gleicher oder ähnlicher Fassung noch nicht in einem anderen Studiengang als Prüfungsleistung vorgelegt habe. Ich habe alle von mir genutzten Hilfsmittel und Quellen, einschließlich generativer Modelle/KI angegeben und die den verwendeten Quellen und Hilfsmitteln wörtlich oder sinngemäß entnommenen Stellen in Form von Zitaten kenntlich gemacht. Darüber hinaus habe ich keine Hilfsmittel verwendet.

Halle (Saale), 23.09.2024

Ort, Datum

Adrian Böschel

Unterschrift

1 Einleitung

1.1 Motivation

Im Rahmen des Moduls *Programmierung eingebetteter Systeme* an der Hochschule Anhalt *Köthen* (HSA) soll ein Projekt bearbeitet werden. In diesem soll in der Programmiersprache Python eine Klasse erstellt werden, die ein MAX4466-Mikrofon/Sensor ausliest. Gedacht ist die Nutzung des Sensors mit der erstellten Klasse als digitales Stethoskop, um Atemgeräusche im Bereich von 200 Hz bis 2500 Hz abzuhorchen [1]. Die entwickelte Klasse soll dann für die zukünftigen Jahrgänge im Modul *Mikrocontrollertechnik* zur Durchführung des Praktikums verwendet werden.

Die Erstellung einer Klasse mit den gegebenen Ansprüchen fordert die fehlerfreie Programmierung und Betrachtung aller Möglichkeiten und Nutzer.

1.2 Ziele

Aus der gegebenen Aufgabenstellung lassen sich folgende umzusetzende Ziele definieren:

- Erstellen einer Klasse in der Programmiersprache MicroPython für die Nutzung des MAX4466 Sensors als digitales Stethoskop
- Mit variabler Abtastfrequenz
- Mit einfachem Funktionsaufruf, sowie Nutzung im Programm
- Funktionalität des Digitalen Stethoskops soll umgesetzt werden
- Einfacher und verständlicher Code für die Nutzung nachfolgender Bearbeiter

2 Theoretische Grundlagen

2.1 Stethoskop

2.1.1 Technologie

Der Begriff „Stethoskop“ kommt vom altgriechischen *stethos* ‚Brust‘ und *skopein* ‚schauen. untersuchen‘. Mithilfe dieses Diagnosewerkzeuges können Schallphänomene im inneren von Hohlkörpern beurteilt werden. Stethoskope werden in der Medizin verwendet, um Töne und Geräusche des Herzens, der Lunge und des Darms abzuhorchen. Bei der sogenannten A*uskultation* können krankhafte Strömungsgeräusche oder Infekte festgestellt werden. Auch im Bereich des Maschinenbaus zur Motoranalyse werden Stethoskope verwendet. Da dieser Beleg sich mit dem Abhören von Lungengeräuschen beschäftigt, beschränkt sich der hier interessante Bereich der Stethoskope auf die medizinische Anwendung [2].

2.1.2 Akustisch

Seit der Entdeckung des Stethoskops im Anfang des 19. Jahrhunderts wurde es weiterentwickelt, bis hin zum heute üblichen Aufbau der in Abbildung 1 dargestellt ist.

Abbildung 1: akustisches Stethoskop, bestehend aus Ohr-Bügel, Schlauch und Kopfstück [2]

Die drei Hauptbestandteile des Stethoskops sind in der Abbildung deutlich zu erkennen. Vom Ohr-Bügel, dem Schlauch bis zum Kopf- oder Bruststück. Das Bruststück spielt dabei eine entscheidende Rolle. Es enthält eine Membran über die die Schwingungen des untersuchten Körpers aufgenommen und über die im Schlauch befindliche Luftsäule weitergegeben werden. Über eine Variation des Drucks vom Kopfstück auf den Körper können unterschiedliche Frequenzbereiche bei der Auskultation verstärkt werden [2].

2.1.3 Elektronisch

Ein elektronisches äquivalent besitzt ein Mikrofon, welches die Aufgabe des Kopfstückes übernimmt und über elektrische Leitungen mit einem Mikrocontroller verbunden ist, welcher die Daten aufzeichnet und verarbeitet. Die elektrische Leitung symbolisieren dabei den Schlauch des akustischen Stethoskops. Der Mikrocontroller steht für den Ohr-Bügel, mit dem die aufgenommenen Schwingungen ausgewertet werden. Die in diesem Beleg verwendete Hardware wird in 2.2 und 2.4 näher betrachtet.

2.2 Raspberry Pi Pico

2.2.1 Mikrocontroller

Das Raspberry Pi Pico Board basiert auf dem *RP2040 microcontroller chip*. Es bildet eine kostengünstige, flexible Entwicklungsplattform, welche viele verschiedene Merkmale beinhaltet. In dem in Abbildung 2 dargestellten Pin-Out Diagramm des Pico's sind dabei einige dieser abgebildet [3].

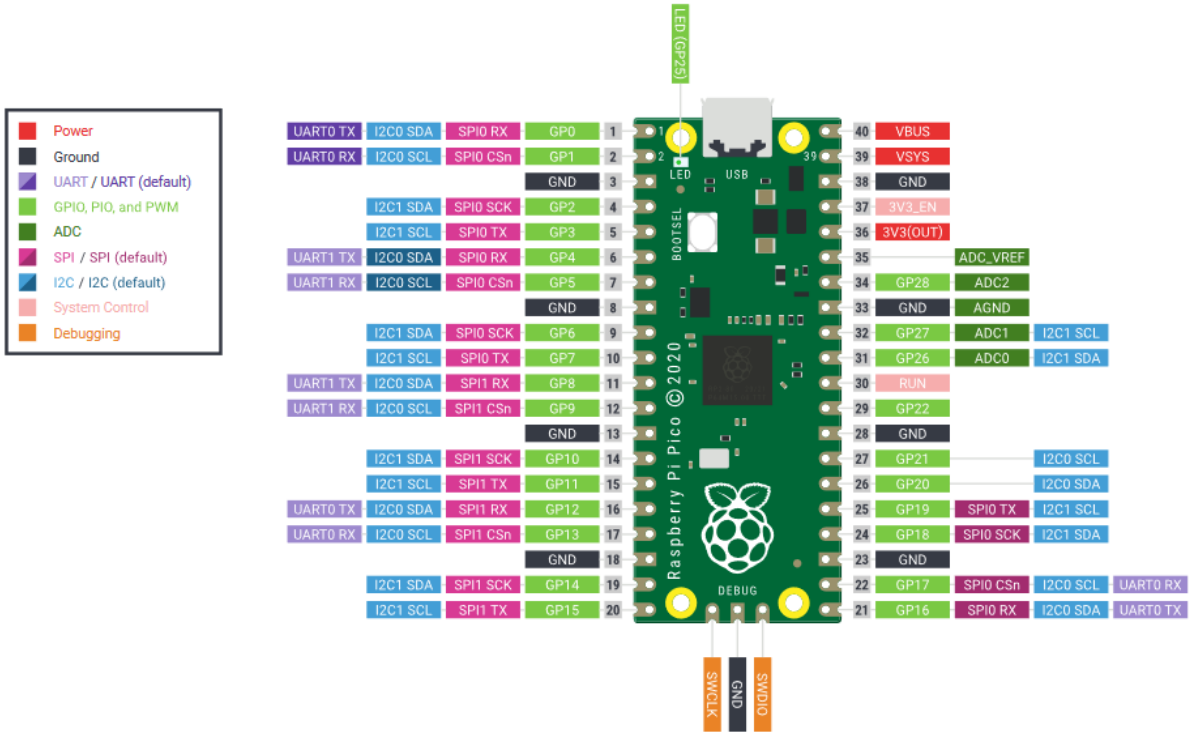


Abbildung 2: Pinout Diagramm des Raspberry Pi Pico [3]

In dem Diagramm sind zum Beispiel der Analog-Digital-Converter an den Pins 26 – 28 des Pico zu erkennen. Diese werden in Abschnitt 2.2.3 genauer betrachtet und sind für die Erstellung eines digitalen Stethoskops essenziell. Unter anderem ist der Pico in der Lage mittels I2C-Busses, UART und SPI zu kommunizieren. Außerdem verfügt er über die Möglichkeit der Verwendung von Puls-Weiten-Modulation (PWM) und einen Timer [3].

Der RaspberryPi Pico benötigt eine Versorgungsspannung von 5V welche an der Micro-USB Steckverbindung angeschlossen wird. Über die Pins VBUS und VSYS können die vom Netzteil zur Verfügung gestellten 5V angeschlossenen Geräten bereitgestellt werden. Der Raspberry Pi Pico selbst nutzt 3.3V als Versorgungsspannung, welche an dem Pin VSYS anliegt [3].

2.2.2 Programmierung

C Die geläufigste Programmiersprache für Mikrocontroller ist C. Auch der Pico wird normalerweise mit dieser Programmiersprache verwendet. Ein Vorteil derer ist besonders die Geschwindigkeit der Programme, jedoch kann es durch falsche Verwendung und Zuweisung des Speichers schnell zu Programmabstürzen kommen. Da diese Besonderheit für unerfahrene Nutzer hinderlich im Lernprozess ist, wurden alternativen für die Programmierung von Mikrocontrollern eingeführt um diese zu erleichtern.

MicroPython Darüber hinaus ist der RP2040 der *Raspberry Pi Foundation* auch mit Micropython programmierbar. Es bietet eine Python ähnliche Sprache, die auch viele der standard Python Bibliotheken unterstützt. Zusätzlich lassen sich die GPIOs, Timer, ADCs und andere auf dem Pico verbaute Merkmale benutzen [4]. Der Vorteil bei der Verwendung von MicroPython liegt besonders in der Lehre bei dem leichteren Umgang mit der Programmiersprache. Da es sich bei Python um eine Interpreter-Sprache handelt, sind die darin erstellten Programme meist nicht so schnell in der Ausführung und erreichen somit eher technische Limitationen. Allerdings zeichnet sich Python in der Lehre besonders durch eine positive Lernerfahrung für die Studenten aus [5].

2.2.3 Analog-Digital-Converter

Ein ADC ist ein Prozess, bei dem ein analoges unendlich präzises Signal abgetastet wird und in ein digitales Signal mit endlicher Präzision umgewandelt wird [6]. In Abbildung 3 ist dieses Prinzip in einer Skizze dargestellt, wobei jede der horizontalen Linien einen möglichen Spannungspegel des ADCs beschreibt. Bei der Quantisierung kann eine Auflösung A abhängig von der Signalamplitude Δ sowie der Bitbreite des ADC ist berechnet werden nach der Formel:

$$A = \frac{Messbereich}{Bitbreite} = \frac{\triangle}{2^n - 1\ Bit}$$

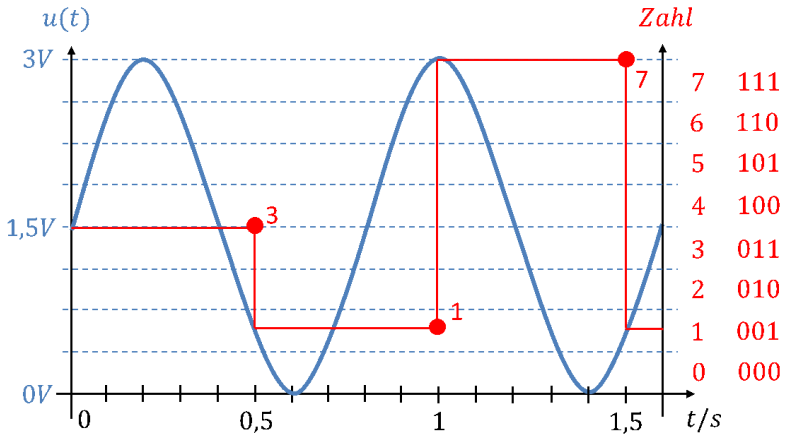


Abbildung 3: Ein 3V-Sinussignals mit einer Periodendauer von 0,8 Sekunden wird mit einem 3-Bit-ADC bei 2 Hertz Abtastfrequenz quantisiert [6]

Für den auf dem Raspberry Pi Pico verbauten 16-Bit-ADC mit einem Messbereich zwischen 0 V bis 3,3 V ergibt sich eine Auflösung von:

$$A = \frac{3{,}3\ V}{2^{16} - 1} = \frac{3{,}3\ V}{65535} = 50{,}35477\ \mu V$$

Die Abtastfrequenz f_A beschreibt dabei, wie viele Werte pro Sekunde aufgenommen werden. Sie soll nach dem Abtasttheorem mindestens doppelt so hoch, wie die maximale Signalfrequenz sein. Wird die Abtastfrequenz nicht hoch genug gewählt, kann das untersuchte Signal nicht vollständig rekonstruiert werden [6].

$$f_A \geq 2 * f_{max}$$

Für die vom Stethoskop aufzunehmenden Atemgeräusche mit einer Frequenz von bis zu 2500 Hz ergibt sich damit eine minimal geforderte Abtastfrequenz von rund 5000 Hz [1].

2.2.4 Timer und Interrupts

Programmablauf Der von einem Mikrocontroller ausgeführte Code wird sequentiell abgearbeitet. Da es jedoch in manchen Fällen wichtig ist, auf ein bestimmtes Ereignis zu reagieren gibt es sogenannte Interrupts.

Interrupts Diese stoppen den aktuellen Programmablauf und führen die dem Ereignis, welches den Interrupt ausgelöst hat, zugehörige *Interrupt Service Routine* (ISR) durch. Nach Beenden der Routine wird dann das Programm an der zuvor unterbrochenen Stelle fortgesetzt. Damit die Unterbrechung im Programmablauf möglichst gering ist, wird empfohlen die ISR so kurz wie möglich zu halten [7].

Timer (Hardware/Software) Für die Ausführung von ISR nach einer gewissen Zeit werden sogenannte Timer verwendet. Der Pico bestitzt einen Timer on-chip [3], welcher mit einem Tick jede Mikrosekunden hochgezählt wird [8]. Der Wert dieses Zählers kann mit bis zu vier

verschiedenen Alarmen belegt werden. Dafür definiert der Nutzer in seinem Programm einen Vergleichswert der bei Übereinstimmung mit dem Counter des Timers dann die ISR auslöst. Da der Takt des Timers direkt aus dem Systemtakt des Quarz-Oszillators gebildet wird, ist der Timer sehr genau und eignet sich für zeitkritische Anwendungen [8].

2.3 Cytron Maker Pi Pico Base

Hierbei handelt es sich um eine Leiterplatte auf die der Pico gesteckt werden kann. Über die 20-poligen Stiftleisten können Verbindungen mittels Überbrückungsdrähten realisiert werden aber auch die Möglichkeit der Nutzung von GROVE-Kabeln ist gegeben. Dadurch wird die flexible Nutzung des Picos vereinfacht und es ist kein Löten notwendig. Weiterhin verfügt das Base Board über die Möglichkeit der Nutzung von LEDS, Tastern, Audio-Buchsen, einem SD-Karten Slot und anderen Technologien [9].

2.4 MAX 4466 / Mikrofon

Definition Bei diesem Mikroleistungsverstärker handelt es sich um einen für den Mikrofoneinsatz optimierten OPV. Dieser kann für *full range audio* genutzt werden [10]. Das bedeutet Signale im Bereich von $20\ Hz$ bis $20\ kHz$ können mit dem Sensor erfasst werden. Damit wird der gesamte Hörbare Tonbereich abgedeckt.

Technische Daten Der OPV hat ein minimales Verstärkungsbandbreiteprodukt von 600 kHz bei einer stabilen Verstärkung von $+5\frac{V}{V}$. Betrieben wird er mit $2.4\ V$ bis $5.5\ V$ Versorgungsspannung, er besitzt eine Slew Rate (SR) von $45\ \frac{mV}{\mu s}$ und ist ein sogenannter Rail-to-Rail-Verstärker [10]. Angeschlossen wird der Sensor an den Pico über drei Verbindungen: VCC, GND und den Signal-Pin.

2.5 Signalanalyse

Ziel dieser Arbeit ist es Erkenntnisse aus den Daten zu gewinnen. Die aufgenommenen Signale im Zeitbereich enthalten Informationen über die Amplitude und den Zeitpunkt der Abtastung. Mit einer höheren Abtastfrequenz lässt sich die gemessene Schallkurve besser rekonstruieren. Mithilfe der *Fast Fourier Transformation* (FFT) ist es möglich die Daten vom Zeitbereich in den Frequenzbereich zu transformieren und somit das Frequenzspektrum zu erhalten. Durch die computergestützte Analyse des Atemgeräusches über längere Zeit, sowie das Zuordnen von Atemnebengeräusche zu bestimmten Zeitpunkten können akustische Phänomene besser erkannt werden [1].

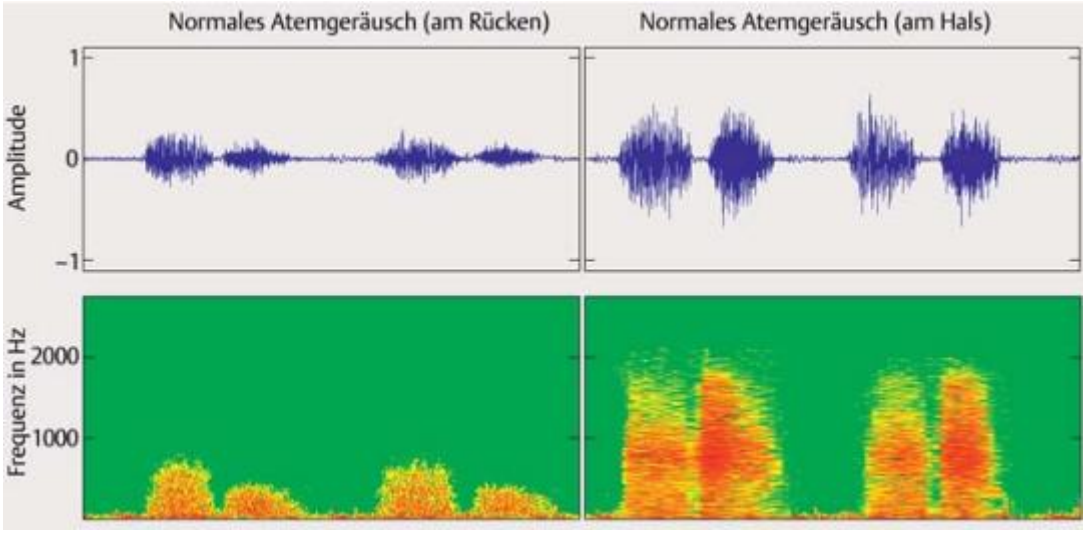


Abbildung 4: Amplituden- (oben) und Frequenzspektrum (unten) eines normalen Atemgeräusches gemessen am Rücken (links) und am Hals (rechts) aufgenommen von Koehler et al. [1]

In Abbildung 4 ist die Aufzeichnung eines Atemgeräusches mit dem dazugehörigen Frequenzspektrum dargestellt. Dabei ist der zeitliche Verlauf klar ersichtlich. Außerdem ist erkennbar, dass der Messpunkt am Patienten eine entscheidende Rolle hinsichtlich der Auflösung des gemessenen Spektrums liefert. Eine Messung am Hals liefert dabei höhere Signalamplituden und auch mehr im Signal enthaltene Frequenzen.

Fast Fourier Transformation Die *diskrete Fourier Transformation* (DFT) ist eine Methode um endliche Signale, deren Punkte gleichmäßig in der Zeit verteilt sind, vom Zeitbereich in den Frequenzbereich zu übertragen. Mathematisch wird die Diskrete Fourier Transformation durch folgende Gleichung beschrieben [11]:

$$X_k = \sum_{m=0}^{n-1} X_m e^{-i2\pi k \frac{m}{n}} \qquad k = 0, \dots, n-1$$

Da die DFT über eine Summation statt eines Integrals gebildet wird, kann diese mit einem Algorithmus berechnet werden. Unter der FFT wird in der Informatik meistens der Algorithmus von Cooley und Tukey [12] verstanden, welcher die Berechnung computergestützt ermöglicht. Dieser sogenannte *Divide and Conquer* Algorithmus zerlegt die ursprüngliche DFT in immer kleiner werdende neue DFTs, bis eine genaue Lösung gefunden werden kann [11].

Der Rekursive Ansatz der FFT hat eine Komplexität von $\sim O(n * \log(n))$ welche eine gute Alternative zur sonst sehr Zeitaufwändigen Berechnung von DFTs darstellt [13].

3 Durchführung

3.1 Hardware Konfiguration

Der Raspberry Pi Pico soll zur Programmierung über den Micro-USB-Stecker verbunden werden. An dem Pico wird dann das Mikrofon (MAX4466) angeschlossen. Dieses verfügt über die Eingänge GND, VCC und SIG. Mit den GND- und VCC-Pins wird dabei die Stromversorgung gewährleistet welche sinngemäß, wie in Abbildung 5 dargestellt, an den Pico angeschlossen werden.

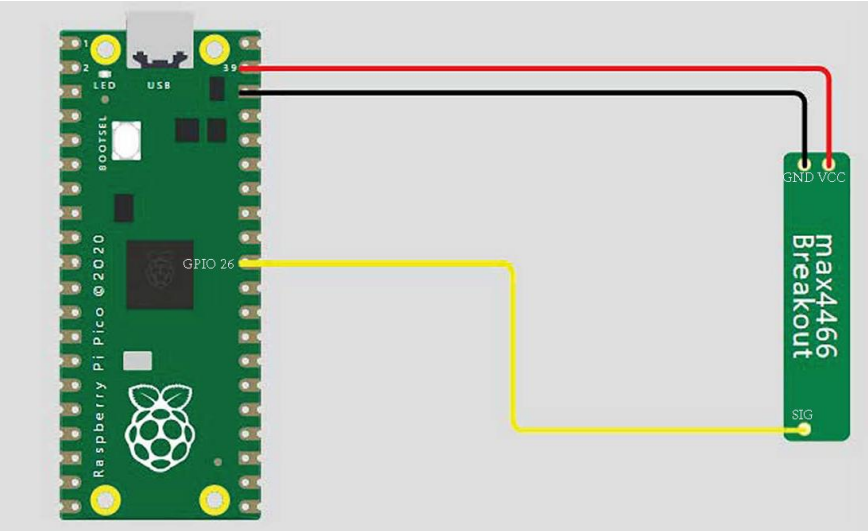


Abbildung 5: Schaltplan des Digitalen Stethoskops, MAX4466-Sensor wird über GND und VCC mit dem Pico verbunden, der Signal-Ausgang des Sensors wird mit einem ADC-Pin des Pico verbunden (hier GPIO26)

Die Signalleitung soll dann mit einem beliebigen ADC-fähigen Pin des Pico verbunden werden. Wie aus Abbildung 2 hervor geht stehen dafür die GPIOs 26 bis 28 zur Verfügung. Der hier gewählte Pin ist wichtig für die Nutzung im Programm. Sollte etwas falsch konfiguriert werden oder ein GPIO ohne ADC Anbindung genutzt, so wird das Programm keine Werte liefern können. Eine Überprüfung der korrekten Konfiguration soll weitestgehend softwareseitig erfolgen. Die Verbindung mit der SD-Karte erfolgt über das Erweiterungsboard. Sie kann mithilfe der Treiberdatei sdcard.py unter der Programmiersprache MicroPython verwendet werden. Der Code hierfür wird von den Entwicklern von MicroPython bereitgestellt [14]. Die mit der SD Karte verbundenen Pins sind dabei durch die interne Verschaltung des Erweiterungsboards vorgegeben. Die zu verwendenden Pins sind in Abbildung 6 dargestellt.

Raspberry Pi Pico GPIO	SD Card	
	SD Mode	SPI Mode
GP10	CLK	SCK
GP11	CMD	SDI
GP12	DAT0	SDO
GP13	DAT1	X
GP14	DAT2	X
GP15	CD/DAT3	CSn

Abbildung 6: Auszug aus der Dokumentation des Erweiterungsboards bezüglich der Verbindung der GPIOs des Pico mit den Anschlüssen des SD-Karten Moduls [9]

3.2 Hauptprogramm

Zuerst wird ein Plan erstellt, wie ein Beispielhaftes auf dem Pico laufendes Main Programm aussehen könnte. Zu Beginn muss die Klasse importiert werden, da die erzeugte Python-Datei, welche die Klasse enthält im */libs* Ordner abgelegt wird. Danach muss die Klasse aufgerufen und das mit der *__init__* Funktion erzeugte Objekt in einer Variable gespeichert werden. Mit dieser kann dann die Funktionalität der zu erstellenden Klasse genutzt werden. Im ersten Programmablauf soll dabei für eine bestimmte Sekundenzeit Werte mit einer bei der Initialisierung festgelegten Frequenz aufgenommen werden.

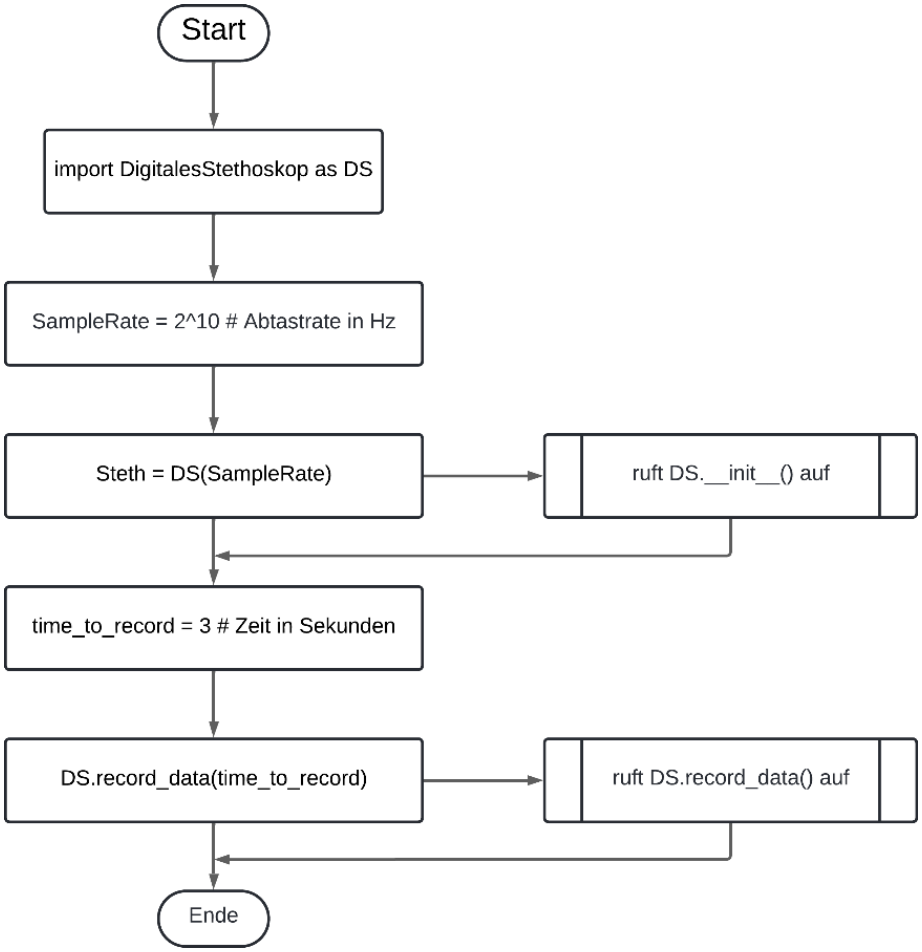


Abbildung 7: Programmablaufplan des Hauptprogrammes um über eine bestimmte Anzahl von Sekunden Werte mit einer festen Abtastfrequenz aufzunehmen

Die vom Hauptprogramm aufgerufenen Funktionen sind in diesem Fall die einzigen notwendigen für die Aufnahme von einer bestimmten Zeitspanne. In Abbildung 5 wurde der MAX4466 mit dem Signal Ausgang an den Pin26 des Picos angeschlossen, da dieser über einen ADC verfügt. Auf dem Development Board ist der Pin außerdem am GROVE6 wiederzufinden, was die Möglichkeit der Nutzung eines GROVE-Steckers oder über Jumperkabel ermöglicht. Aufgrund dieser vielseitigen Verwendbarkeit des Pin26 wird dieser standardmäßig als Signal-Pin festgelegt, kann jedoch bei der Initialisierung zu einem anderen ADC fähigen Pin geändert werden.

3.3 Erstellen der Klasse

3.3.1 Attribute und Methoden

Attribute Sie sind maßgeblich für die Konfiguration und Initialisierung der Klasse. Mit Ihnen werden die Parameter bestimmt unter denen die Aufzeichnung der Messwerte stattfindet. Für die Konfiguration der Klasse wichtige Attribute sind in Tabelle 1 zusammengefasst. Diese können bei der Initialisierung der Klasse direkt übergeben werden oder später konfiguriert.

Tabelle 1: Zusammenfassung der Attribute, welche das Verhalten der Klasse definieren

Nr.	Attribut	Optionale Argument	Daten-typ	Beschreibung
1	SAMPLE_RATE	Nein	int	Die Abtastrate mit der die Werte später aufgezeichnet werden
2	SIGNAL_PIN	Ja	Pin	Machine.Pin-Objekt an dem der MAX4466 Sensor mit der Datenleitung angeschlossen ist; standardmäßig Pin(26)
3	MAX_DATA	Ja	int	Beschreibt die maximale Anzahl an Werten, die aufgenommen werden darf um ein überfüllen des RAM-Speichers zu vermeiden; standardmäßig 8000
4	DATEI_NAME	Ja	str	Name der Datei unter dem die Werte gespeichert werden, mit „.txt“ oder „.csv“ am Ende!; standardmäßig „data.txt“

Methoden Die in einer Klasse definierten Funktionen nennen sich Methoden, da sie über das Klassenobjekt aufgerufen werden können und Zugriff auf die dem Objekt eigenen Variablen hat. Die Methoden der Klasse des Digitalen Stethoskops sind in Tabelle 2 aufgeführt und beschrieben.

Tabelle 2: Übersicht aller öffentlichen für den Nutzer sinnvollen Methoden

Nr	Methode	Beschreibung
1	record_data(time_in_s:int)	Startet eine Werteaufnahme über einen bestimmten Zeitraum in Sekunden in dem der Pico keine anderen Aktionen ausführen kann, um ein möglichst genaues Ergebnis zu erzielen;
2	save(path=self.PATH)	Speichert die aufgezeichneten Werte in einer Datei auf dem Flash-Speicher des Pico ab
3	Save_sd(path=self.PATH)	Speichert die aufgezeichneten Werte auf der SD-Karte ab, nur bei erfolgreicher Initialisierung der SD-Karte möglich
4	get_data()	Gibt alle aktuell im RAM-Speicher vorhandenen Daten als Array zurück, sollten keine Daten aufgezeichnet worden sein gibt es eine entsprechende Meldung

3.4 Programmablaufplan

Der in Abbildung 7 dargestellte Ablaufplan des Hauptprogrammes ist für die Aufnahme von Werten über einen bestimmten Zeitraum verantwortlich. Im Folgenden werden die darin aufgerufenen Funktionen und deren Ablauf beschrieben.

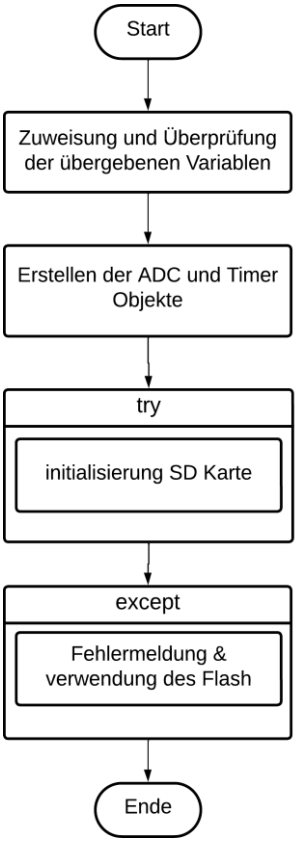


Abbildung 8: Programmablaufplan der *__init__* Methode der Klasse

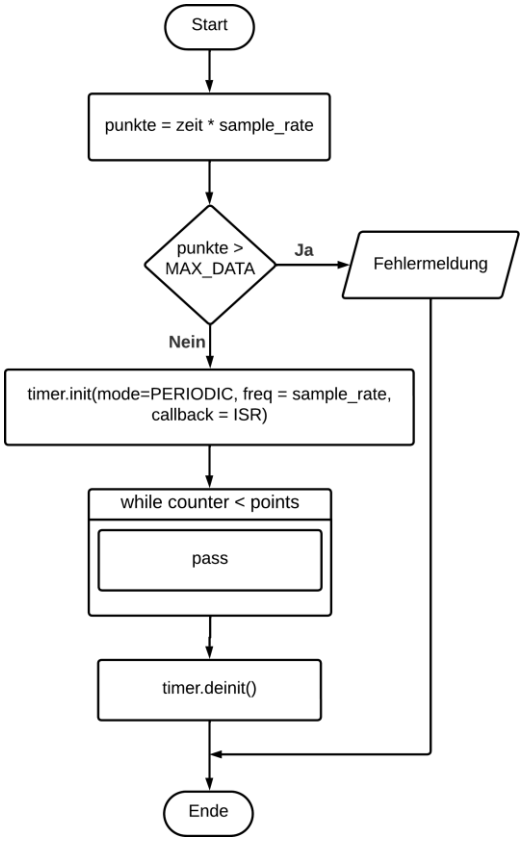


Abbildung 9: Programmablaufplan der *record_data* Methode um Daten über einen bestimmten Zeitraum aufzunehmen und in einer Datei zu speichern

Die *__init__* Funktion wird bei Erstellung der Klassenvariable automatisch ausgeführt und beschäftigt sich zum Großteil mit dem Anlegen von Variablen und der Konfiguration der Klasse. Hierbei soll immer die gewünschte Abtastfrequenz übergeben werden, während alle anderen Variablen optional sind. Nach der Zuweisung und Überprüfung der übergebenen Werte wird dann die SD-Karte initialisiert, sollte dies fehlschlagen oder keine Karte verwendet werden kann lediglich auf den Flash-Speicher geschrieben werden.

In der *record_data* Funktion selbst werden keine Werte aufgezeichnet, sondern lediglich der Timer gestartet, welcher die ISR auslöst. Zuvor wird überprüft, ob die Menge an aufzuzeichnenden Daten auch in den Verfügbaren Speicher passt. Anschließend wird so lange gewartet, bis alle Werte aufgenommen wurden und dann die Funktion beendet. Ein PAP der Werteaufzeichnung in der ISR ist in Abbildung 10 dargestellt. Sie soll wie in 2.2.4 beschrieben möglichst kurz gehalten werden, um Fehler im Programmablauf zu vermeiden.

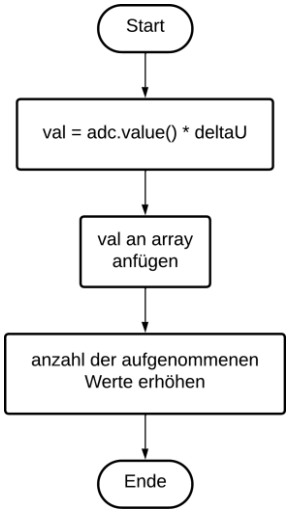


Abbildung 10: PAP der ISR für das Speichern eines Wertes im RAM Speicher des Pico

Die in der Methode durchgeführten Anweisungen sind es, den ADC Wert auszulesen und zu verrechnen ihn anschließend im RAM-Speicher abzulegen und die Anzahl der gespeicherten Werte zu erhöhen. Mit diesem kleinstmöglichen Ablauf der ISR wird garantiert, dass selbst bei hohen Frequenzen die Routine noch vollständig und problemlos durchgeführt werden kann. Die auf dem RaspberryPi Pico vorhandenen 264 KB RAM-Speichers [3] sind für die kontinuierliche Aufzeichnung von Daten bei hohen Abtastfrequenzen sehr schnell gefüllt. Um den zur Verfügung stehenden Speicher nicht zu überschreiten wird die Variable MAX_DATA genutzt. Sie limitiert die maximale Anzahl vom Pico bereitgehaltener Daten.

Ein Speichern der Daten für die Auswertung und den Erhalt der Daten kann dann auf dem Flash-Speicher des Pico oder einer SD Karte geschehen. Da dieser verhältnismäßig langsam ist eignet er sich aber nicht zur Aufnahme von Daten in Echtzeit.

3.5 Test des Systems

3.5.1 Bewertung

Um die Qualität und Güte des gemessenen Signals zu analysieren werden verschiedene Versuche durchgeführt. Dabei wird das Signal mit verschiedenen Abtastfrequenzen und über einen bestimmten Zeitraum aufgenommen werden. Das Amplituden- und Frequenzspektrum jeder Aufnahme soll graphisch dargestellt und anschließend analysiert werden.

3.5.2 Versuch 1: Signalgenerator

Im ersten Versuch soll mithilfe eines Signalgenerators ein reines Sinussignal an den ADC-Pin des RaspberryPis angelegt werden. Dies dient insbesondere der Überprüfung des Algorithmus hinsichtlich der zeitlichen Genauigkeit. Zeitsprünge in der Abtastung sollten hier sofort sichtbar werden und auch andere mögliche Fehler erkenntlich.

Die Amplitudenwerte des Sinussignals ergeben sich aus den für den ADC vorgegebenen Richtlinien an den Spannungsbereich des angelegten Signals [8]. Es wird erwartet, dass dieses Signal gut rekonstruiert werden kann und in der Frequenzanalyse eindeutig die im Signal enthaltene Frequenz wiederzufinden ist. Es werden drei sinusförmige Signale mit jeweils drei verschiedenen Abtastraten aufgezeichnet. Aufgrund dessen kann eine Übersicht gebildet werden hinsichtlich der Qualität der Abtastung und der Frequenzanalyse erstellt werden. In Tabelle 3 sind die am Signalgenerator einzustellenden Kennwerte aufgeführt.

Tabelle 3: Kennwerte der drei abzutastenden Sinussignale die am Frequenzgenerator eingestellt werden

Nr	Signalfrequenz	Spitze-Spitze-Spannung	Signaloffset
1	$f_s = 45\ Hz$		
2	$f_s = 450\ Hz$	$U_{SS} = 3\ V$	$U_{Offset} = 1{,}5\ V$
3	$f_s = \frac{f_a}{2} - 50Hz$		

Die Einstellung des 450 Hz Sinussignals mit den für den ADC verträglichen Kennwerten am Frequenzgenerator ist dabei in Abbildung 11 dargestellt, während der Versuchsaufbau mit der Verbindung zwischen Frequenzgenerator und Pico in Abbildung 12 zu sehen ist.

Abbildung 11: Auf Kanal A eines Frequenzgenerators eingestelltes Sinussignal Nummer 2 aus Tabelle 3, mit einer Frequenz von 450 Hz einer Spitze-Spitze-Spannung von 3 Volt und einem 1,5 Volt Offset

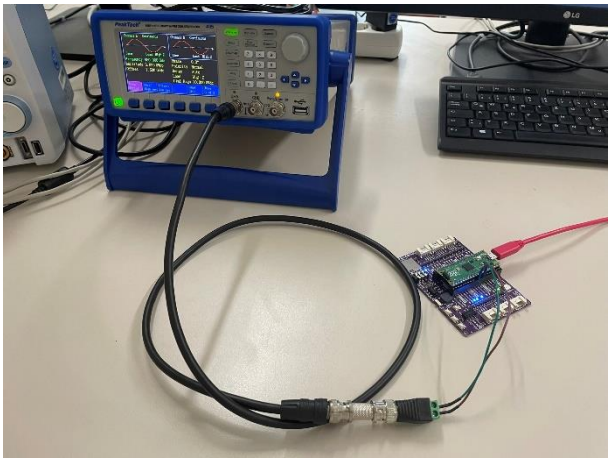


Abbildung 12: Versuchsaufbau zur Verwendung des Funktionsgenerators, dessen Ausgang an GPIO26 und GND des Pico angeschlossen wurden

3.5.3 Versuch 2: Ton mit fester Frequenz

Darauf aufbauend soll der MAX4466 Sensor wie geplant als Mikrofon verwendet werden. Mithilfe eines mobilen Endgerätes wird in der Nähe ein Ton mit konstanter Frequenz wie in 3.5.2 erzeugt und aufgezeichnet.

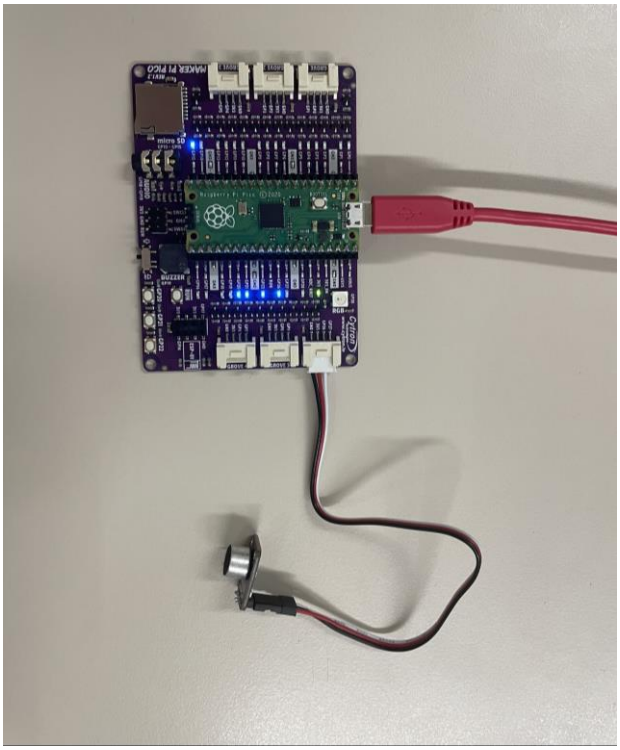


Abbildung 13: Versuchsaufbau von Versuch 2, Anschluss des Pico mittels GROVE Connector 6 an dem GPIO26 und Verbindung mit PC über microUSB

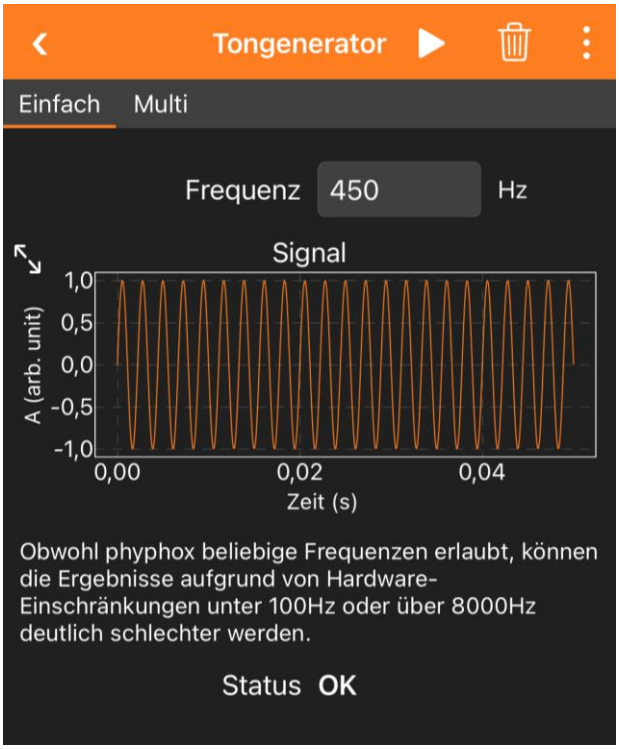


Abbildung 14: Mit der Anwendung Pyphox erzeugter Ton von 450 Hz auf einem iPhone 12 mini welches in der Nähe des Picos den Ton abspielt

Der hierbei verwendete Versuchsaufbau ist in Abbildung 13 dargestellt und wurde wie in 3.1 beschrieben durchgeführt. Der Ton mit konstanter Frequenz wurde mit einer kostenlosen Anwendung auf einem mobilen Endgerät erzeugt. Ein Screenshot der Anwendung ist in Abbildung 14 zu sehen.

3.5.4 Versuch 3: Atemgeräusch

Anschließend wird der MAX4466 an den Hals der Testperson gehalten und eine Aufzeichnung gestartet, während der Proband für die gesamte Dauer normal atmet. Der Abtastpunkt wurde am Hals gewählt, da die dort erwarteten Ergebnisse mehr Frequenzanteile enthalten sollten als bei einer Messung am Rücken [1]. Aufgrund der geringen Zeitdauer in der mit einer hohen Abtastrate aufgenommen werden kann, wird der Vorgang des Ein- und Ausatmens getrennt aufgezeichnet. Der Versuchsaufbau bleibt hier genauso wie in Versuch 2.

3.5.5 Auswertung

Die Darstellung der Amplituden- und Frequenzspektren wird mithilfe der *matplotlib* durchgeführt. Dabei wird das Signal im Zeitbereich so dargestellt, wie es aufgezeichnet wurde. Die dabei zu verwendenden Zeitabstände lassen sich über die Abtastfrequenz bestimmen. Liegt eine nicht kontinuierliche Aufzeichnung vor, sollte diese besonders in Versuch 1 ersichtlich werden. Für die Darstellung des Frequenzspektrums müssen die Daten erst in den Bildbereich transformiert werden. Dafür bietet sich die Verwendung des Python Moduls *numpy* an, da diese den FFT-Algorithmus bereitstellt. Der für die Auswertung und Erstellung der Diagramme genutzte Code ist im Anhang A zu finden.

4 Auswertung

4.1 Versuchsauswertung

4.1.1 Versuch 1: Signalgenerator

Zeitbereich Zum Vergleich der Abtastungen von Signalen mit verschiedenen Signalen ist die komplette Darstellung des Sinussignals wie in Abbildung 15 zu sehen nicht sehr aussagekräftig. Aufgrund dessen werden in der folgenden Auswertung lediglich die ersten 200 Punkte eines Signals im Zeitbereich betrachtet, was Beispielhaft in Abbildung 16 dargestellt ist. Die Abbildungen aller Signale sind im Anhang B zu finden.

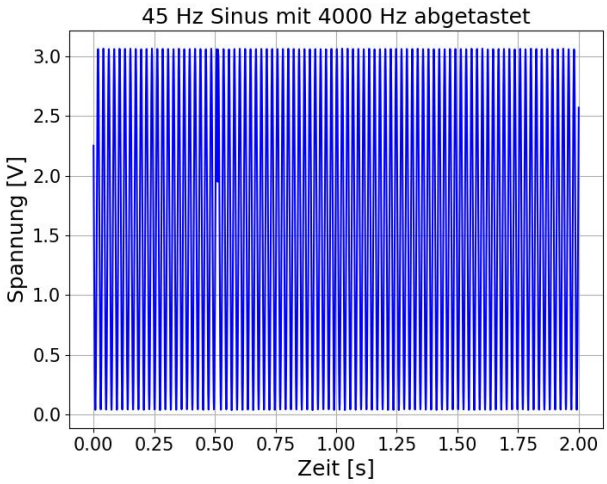


Abbildung 15: komplette Darstellung eines 45Hz Sinus, welcher über eine Dauer von 2 Sekunden mit 4000 Hz abgetastet wurde

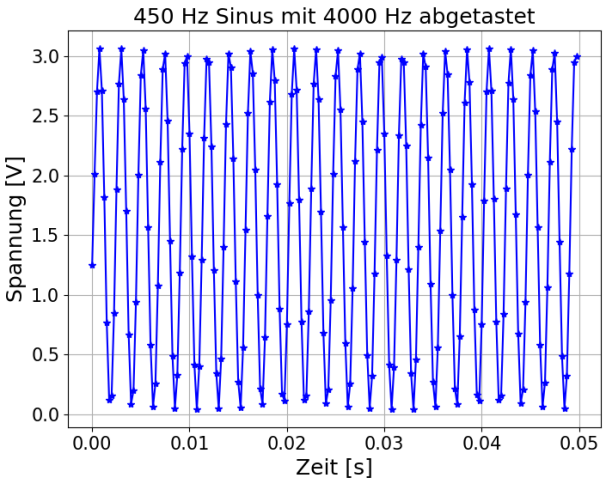


Abbildung 16: Darstellung der ersten 200 aufgezeichneten Punkte des in Abbildung 15 gezeigten Sinussignals

In Abbildung 15 ist besonders deutlich bei 0.5 Sekunden eine Unterbrechung der Aufzeichnung zu erkennen, da dort nicht kontinuierlich das Sinussignal aufgenommen werden konnte. Dieser sogenannte Zeitsprung ist in Abbildung 17 noch einmal exemplarisch dargestellt. In der im folgenden Abschnitt „Frequenzbereich“ durchgeführten Analyse wird die Auswirkung eines solchen Zeitsprungs im Frequenzbereich betrachtet.

Zeitsprünge In der MikroPython Dokumentation sind die Limitierungen der Timer bei Verwendung eines RP2040 nur soweit beschrieben, dass lediglich Software Timer verwendet werden können. Deren Limitationen äußern sich besonders in der geringen Priorität der Programmdurchführung, in denen auch das USB Protokoll oder andere Interrupts die eigentlich eigene ISR unterbrechen können. Die Geschwindigkeit und der Ablauf der Software Timer wurde mithilfe eines Oszilloskops untersucht.

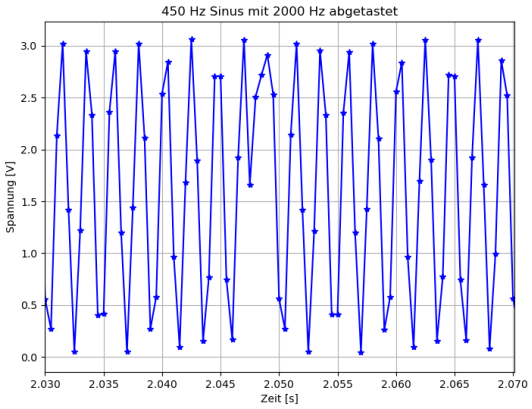


Abbildung 17: Ausschnitt eines Zeitsprungs in der Aufzeichnung durch Pausen in der Abtastung eines Sinussignals

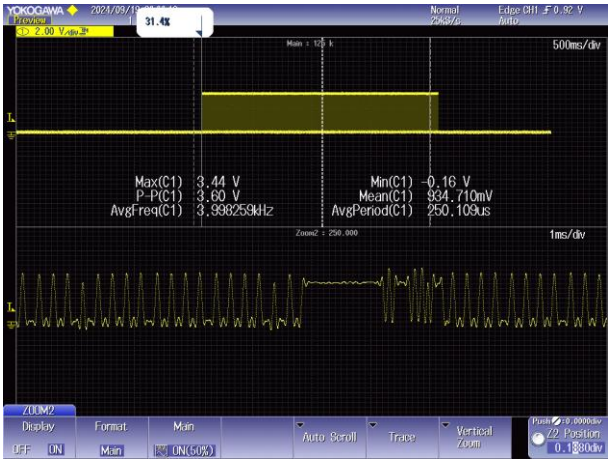


Abbildung 18: Am Oszilloskop aufgezeichnete Unterbrechung in der Aufnahme in der Interrupt Service Routine; 4000Hz als Abtastfrequenz gewählt

Dafür wurde das Messgerät an einem Pin des Pico angeschlossen und dieser in der ISR vor dem aufzeichnen eines Wertes auf HIGH gesetzt und nach Abschluss in der ISR wieder auf LOW gesetzt. Somit kann die allgemeine Dauer einer ISR sowie die Frequenzgenauigkeit der Abtastrate untersucht werden. In Abbildung 18 wurde dabei eine ISR die mit vier Kilohertz aufzeichnet mit dem Oszilloskop untersucht und ein Zeitbereich vergrößert in dem eine solche Unterbrechung der ISR zu erkennen war. Mit der Messfunktion konnte allerdings gezeigt werden, dass die durchschnittliche Schaltfrequenz sehr nah an den vier Kilohertz liegt und auch die Gesamtdauer der Abtastung den gewünschten zwei Sekunden entspricht. Beobachtet wurde außerdem, dass die Unterbrechungen bei jeder Durchführung auftreten, jedoch in der Dauer ohne erkennbaren Zusammenhang variieren können. Im Anhang G sind weitere aufgenommene ISR mit verschiedenen Frequenzen abgebildet. Der in der ISR ausgeführte Code zum toggeln des Pins ist in Anhang E hinterlegt.

Frequenzbereich Die komplette Übersicht der mittels FFT berechneten Frequenzspektren der abgetasteten Signale ist in Anhang C dargestellt. Es konnten in allen Fällen die im Signal enthaltenen Frequenzen deutlich erkannt werden. Sollte ein größerer Zeitsprung im Signal vorhanden sein, so wird das Frequenzspektrum deutlich verrauschter und hat viele höhere Frequenzanteile aufgrund der schnellen Änderung des Signals im Vergleich zur konstanten Frequenz des Sinus wie besonders in Abbildung 19 klar zu erkennen. Das nur mit wenigen hochfrequenten Anteilen versehene Frequenzspektrum eines 450 Hz Sinus ist in Abbildung 20 dargestellt, was auf weniger gravierende Unterbrechungen der ISR bei der Werteaufnahme rückschließen lässt.

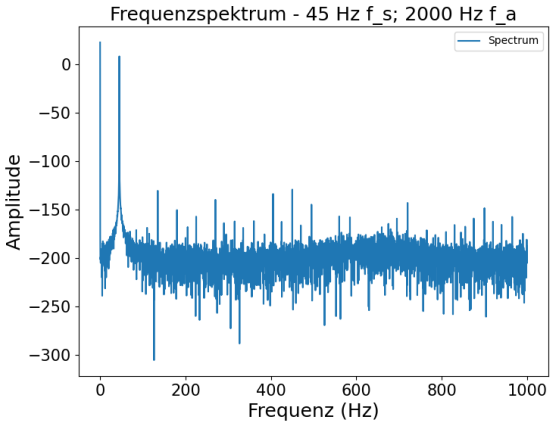


Abbildung 19: Frequenzspektrum eines 45 Hz Sinus der mit einer Abtastfrequenz von 2000 Hz aufgezeichnet wurde, aufgrund vieler Zeitsprünge gibt es hier besonders im Hochfrequenten Bereich viele kleinere Peaks

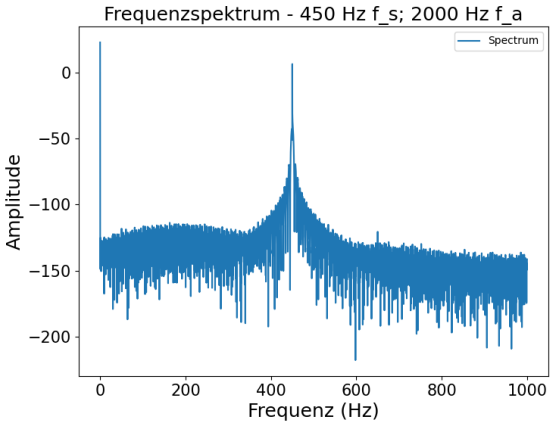


Abbildung 20: Frequenzspektrum eines 450 Hz Sinus mit einer Abtastfrequenz von 2000 Hz aufgezeichnet; wegen weniger langen Unterbrechungen in der Aufzeichnung ist das Frequenzspektrum deutlich glatter

Spektrogramm Zusätzlich gibt es die Möglichkeit den Verlauf der im Signal enthaltenen Frequenzen über die Zeit darzustellen. In dem Spektrogramm werden dabei die Zeitsprünge über Spitzen im gesamten Frequenzbereich zu einem Zeitpunkt deutlich. In Abbildung 21 wird dabei der Zeitsprung bei Sekunde eins und zwei und kurz vor Ende deutlich. Im restlichen Verlauf der Werteaufnahme sind die 450 Hz klar die einzige im Signal enthaltene Frequenz, was den erwarteten Werten entspricht. Eine Übersicht der Spektrogramme befindet sich in Anhang D.

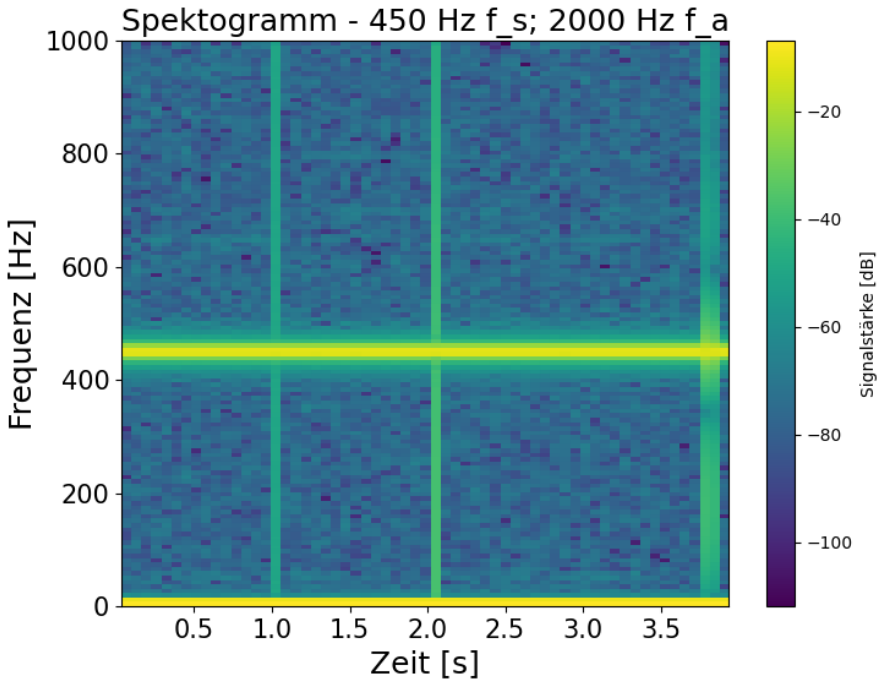


Abbildung 21: Spektrogramm des 450 Hz Sinussignals mit 2000 Hz Abtastrate aufgenommen über einen Zeitraum von vier Sekunden

Ergebnis Zusammenfassend kann die Aussage getroffen werden, dass die Zeitsprünge die Aufzeichnung stören, jedoch die Ergebnisse trotz allem ausreichend genau sind um ein gut analysierbares Ergebnis zu erzielen. Die größere Limitierung ist dabei der geringe verfügbare Arbeitsspeicher des Pico. Da die Werte mit einer hohen Geschwindigkeit aufgezeichnet werden muss der Speicher diese auch schnell genug aufnehmen können. Mit dem integrierten Flash Speicher ist das nicht möglich, wodurch lediglich der 256 kByte große RAM-Speicher genutzt werden kann und somit die Dauer der Aufzeichnung besonders für hohe Abtastraten stark limitiert.

Es wurde festgestellt, dass nicht mehr als 8000 Werte auf einmal aufgezeichnet werden können und dies wurde auch in der erstellten Klasse limitiert, um Speicher Fehler im Programmablauf zu verhindern. Eine Veränderung dieser Variable wird nicht empfohlen.

4.1.2 Versuch 2: Ton mit fester Frequenz

Wie in 0 beschrieben wurde der Versuch durchgeführt. Dabei wurde ein 450Hz-Ton abgespielt und mit den drei im ersten Versuch genutzten Frequenzen abgetastet. Aus den gewonnenen Daten konnten dann die folgenden Abbildungen konstruiert werden.

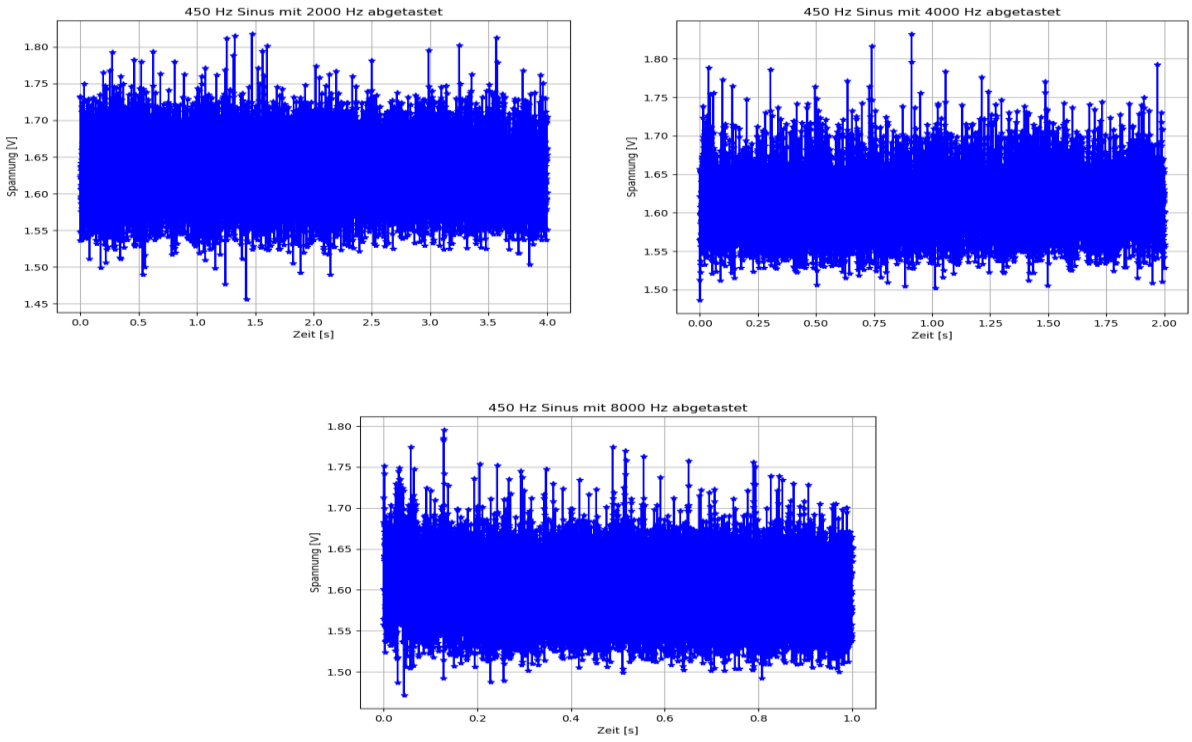


Abbildung 22: Zeitverläufe eines 450Hz Sinus Tons erzeugt mit einem mobilen Endgerät und verschiedenen Abtastraten aufgezeichnet, die Dauer der Abtastung ist dabei durch die technischen Limitationen beschränkt

Auf Basis der mit dem MAX4466 Sensor angeschlossenen Daten wurde dann wieder über die FFT eine Auswertung durchgeführt. Diese sind in Abbildung 23 dargestellt.

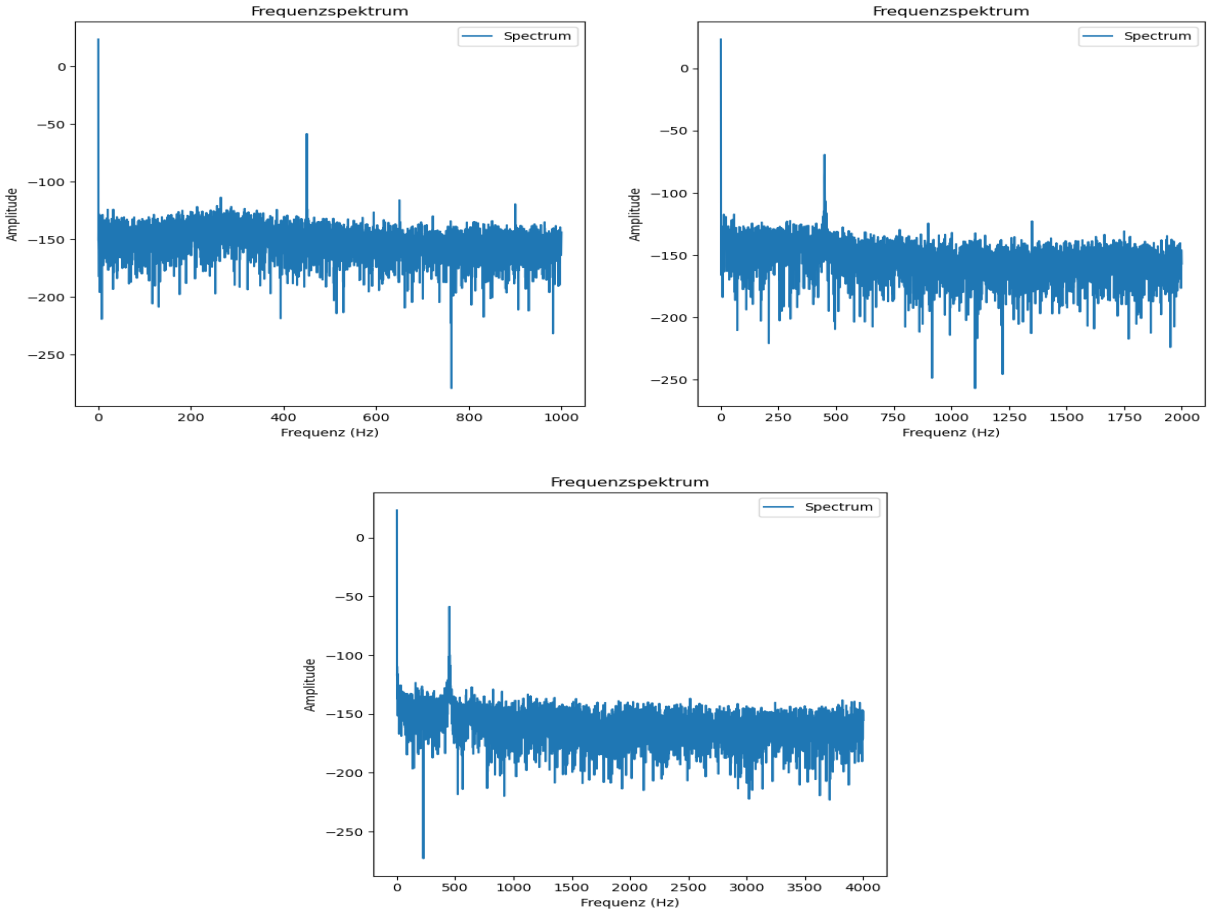


Abbildung 23: Frequenzspektren der Abbildung 22 aufgezeichneten Daten mit deutlichen Peaks der gesuchten Frequenzen

Auch in diesem Versuch konnten geringe Hochfrequente Anteile wegen der Zeitsprünge gemessen werden. Diese sind in den Frequenzspektren gut erkennbar. Aufgrund des höheren Rauschens bei der Verwendung des Sensors sowie möglicher Hintergrundgeräusche ist das im Signal enthaltene Rauschen deutlich stärker, wodurch jedoch auch die Auswirkungen der Zeitsprünge nicht so deutlich sind. Es konnte aber mit jeder Abtastfrequenz die gegebene Frequenz ermittelt werden.

4.1.3 Versuch 3: Atemgeräusch

Zeitbereich Aus der wie in 3.5.4 beschriebenen Durchführung folgte im Zeitbereich bei einer Abtastung mit geringer Frequenz über einen langen Zeitraum ein klarer Ausschlag der Signalamplitude zum Zeitpunkt des Atemzugs. Dies ist in Abbildung 24 bei 0,5 Sekunden klar zu erkennbar. Bei einer hohen Abtastfrequenz, wie in Abbildung 25 dargestellt, kann aufgrund der kurzen Dauer der Aufzeichnung nicht gewartet werden bis der Atemzug startet und das Signal weist somit durchgehend höhere Amplituden auf. Eine Übersicht aller gemessenen Ein- und Ausatemsignale befindet sich in Anhang F.

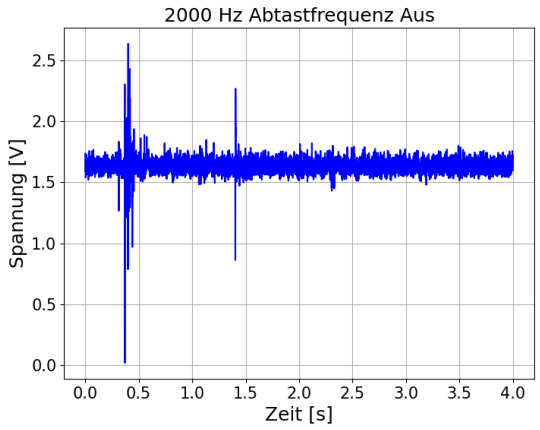


Abbildung 24: Aufnahme eines Ausatemzugs der mit 2000 Hz über vier Sekunden abgetastet wurde

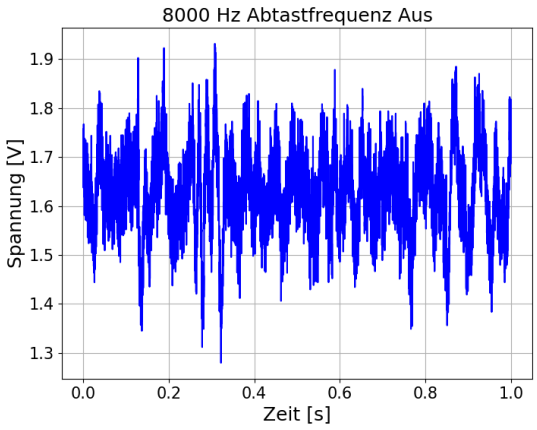


Abbildung 25: Aufnahme eines Ausatemzugs der mit 8000 Hz über eine Sekunde aufgezeichnet wurde

Frequenzbereich Die Frequenzanalyse der Messreihen ergab, dass Atemgeräusche die mit niedrigen Frequenzen abgetastet wurden keine genauen Ergebnisse liefern. In Abbildung 26 sind niedrige Frequenzen von <100 Hz klar zu erkennen und es gibt einen leichten Anstieg bei ca. 400 Hz. Jedoch ist in Abbildung 27 klar zu erkennen, dass die höheren Frequenzen in der Amplitude deutlich dominieren. Bei der Abtastung mit 4000 Hz konnten somit Frequenzen im sehr niedrigen Bereich von <100 Hz sowie bei ca 1400 Hz nachgewiesen werden. Dieses Ergebnis bestätigt sich noch einmal bei der Abtastung mit 8000 Hz, wie in Anhang F zu sehen.

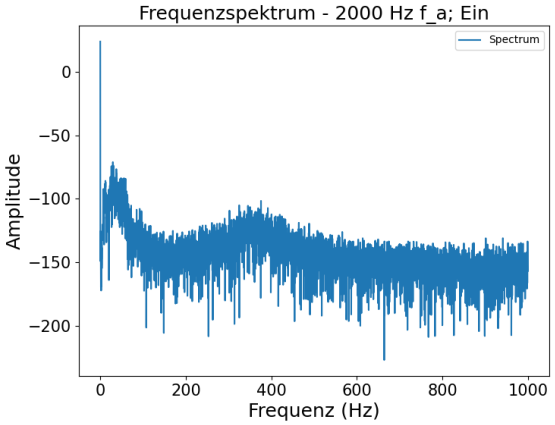


Abbildung 26: Aufnahme eines Einatemzugs der mit 2000 Hz abgetastet wurde

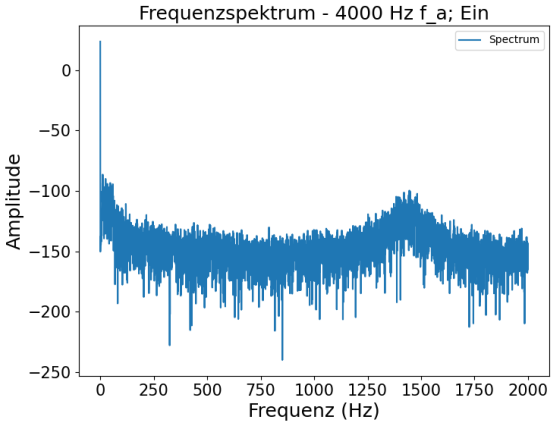


Abbildung 27: Aufnahme eines Einatemzugs der mit 4000 Hz abgetastet wurde

Ergebnis Die Analyse der in Atemgeräuschen enthaltenen Frequenzen ist also mit der erstellten Klasse möglich. Eine Wahl der Abtastrate von 8000 Hz oder höher ist nicht sinnvoll, sofern es sich um einen gesunden Menschen handelt. Bei Verdacht auf Pfeiftöne in der Atmung könnte eine Analyse mit so hohen Frequenzen durchaus Sinn ergeben, sonst ist es von Vorteil einen etwas längeren Zeitraum betrachten zu können. Von den hier gewählten Frequenzen war deshalb die Abtastung mit 4000 Hz die geeignetste.

4.2 Analyse der erstellten Klasse

Ein wichtiger Punkt der Programmerstellung war die einfache Verwendung der Klasse, sowie deren Dokumentation. Mit der Nutzung von Doc-Strings und Kommentaren wurde dies im Code realisiert, während die geplante Anwendung der Funktionen im Rahmen dieser Arbeit beschrieben wurde. Damit wurde das möglichst leicht verständliche und für die Lehre nutzbare Projekt erfolgreich umgesetzt.

5 Literaturverzeichnis

[1] O. H. e. a. U. Koehler, „Atemgeräusche und Atem-Nebengeräusche - Nomenklatur und visuelle Darstellung," in *Pneumologie*, Stuttgart, Georg Thieme Verlag KG, 2016, pp. 397-404.

[2] „Stethoskop," Wikipedia, 20 Juli 2024. [Online]. Available: https://de.wikipedia.org/wiki/Stethoskop#Akustische_Stethoskope. [Zugriff am 25 Juli 2024].

[3] „Raspberry Pi Pico Datasheet," Raspberry Pi Ltd, 2 Mai 2024. [Online]. Available: https://datasheets.raspberrypi.com/pico/pico-datasheet.pdf. [Zugriff am 5 Juli 2024].

[4] P. S. e. a. Damien P. George, „The MicroPython Documentation," [Online]. Available: https://docs.micropython.org/en/latest/rp2/quickref.html. [Zugriff am 25 Juli 2024].

[5] H. Fangohr, „A Comparison of C, MATLAB, and Python as Teaching Languages in Engineering," in *Computational Science - ICCS 2004*, Springer, 2004, pp. 1210-1217.

[6] R. Nocker, Digitale Kommunikationssysteme 1 Grundlagen der Basisband-Übertragungstechnik, Wiesbaden: Vieweg Verlag, 2004.

[7] „Interrupt," mikrocontroller.net, [Online]. Available: https://www.mikrocontroller.net/articles/Interrupt. [Zugriff am 21 September 2024].

[8] „RP2040 Datasheet," Raspberry Pi Ltd, 2 Mai 2024. [Online]. Available: https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf. [Zugriff am 12 September 2024].

[9] C. Technologies, „Maker Pi Pico Datasheet," März 2021. [Online]. Available: https://docs.google.com/document/d/1JoHsZk5IipQPCLXWbZYpDKjGlnkyACOJ1taUrKVsRg8/edit?pli=1. [Zugriff am 12 September 2024].

[10] „Low-Cost, Micropower, SC70/SOT23-8, Microphone Preamplifiers with Complete Shutdown," Maxim, Juni 2012. [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/MAX4465-

MAX4469.pdf. [Zugriff am 5 Juli 2024].

[11] „Fast Fourier Transform," Wikipedia, 9 September 2024. [Online]. Available: https://en.wikipedia.org/wiki/Fast_Fourier_transform. [Zugriff am 13 September 2024].

[12] J. W. T. James W. Cooley, „An Algorithm for the Machine Calculation of Complex Fourier Series," *JSTOR,* pp. 297-301, 1965.

[13] J. Schloss, „What Makes a Fourier Transform Fast?," Algorithm Archive, [Online]. Available: https://www.algorithm-archive.org/contents/cooley_tukey/cooley_tukey.html. [Zugriff am 18 September 2024].

[14] MicroPython, „RAW sdcard.py," [Online]. Available: https://raw.githubusercontent.com/micropython/micropython-lib/refs/heads/master/micropython/drivers/storage/sdcard/sdcard.py. [Zugriff am 18 September 2024].

[15] L. J. J. Tan, Digital Signal Processing - Fundamentals and Applications (3rd Edition), Elsevier, 2019.

[16] Glasmacher, „Abtastung," [Online]. Available: https://www.professorglasmachers.de/mt_abtastung/. [Zugriff am 29 Juni 2024].

Anhang

Anhang A

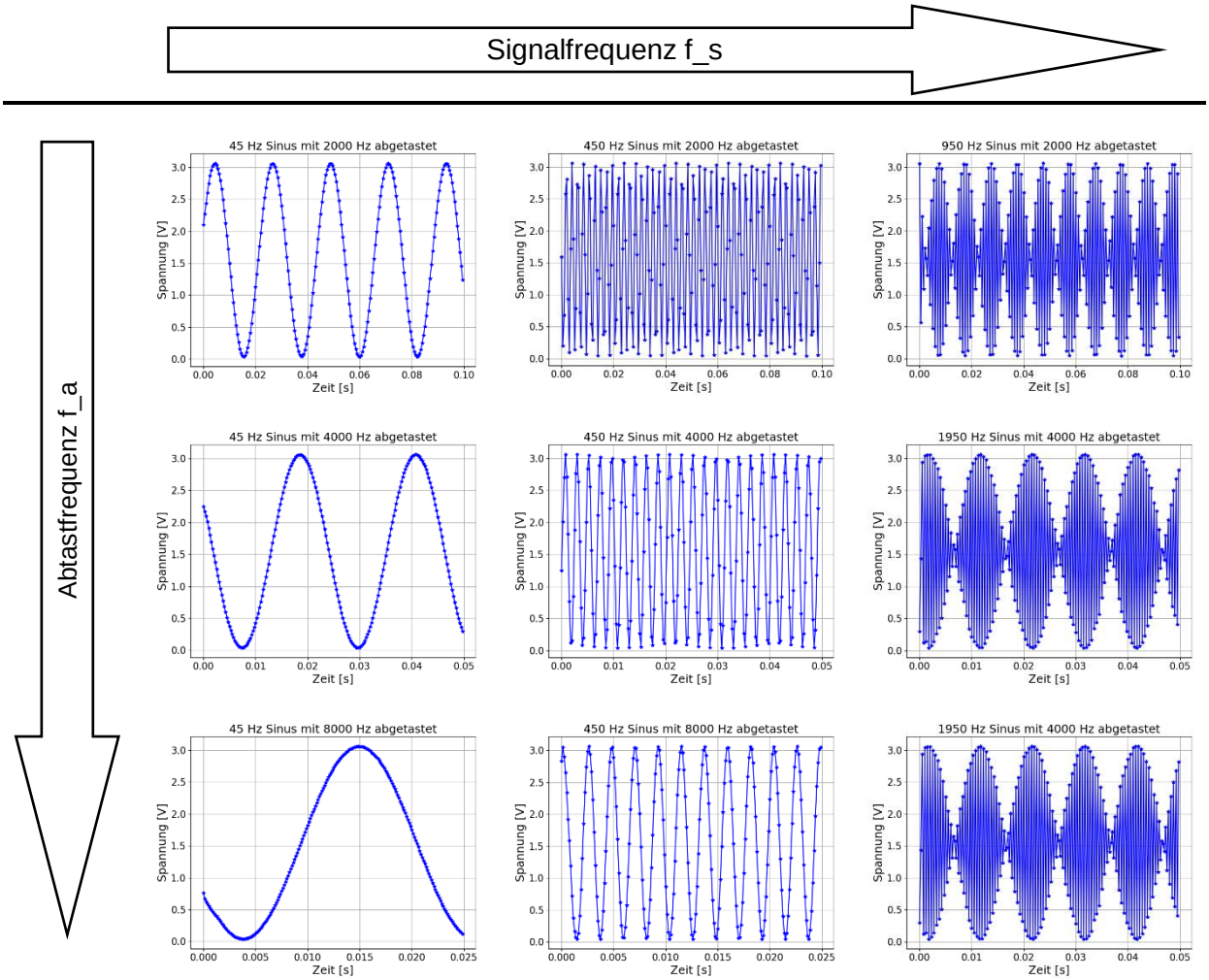
Code zur Auswertung der aufgezeichneten Daten

```
import os, csv, math
import numpy as np
import matplotlib.pyplot as plt

font = 18
ticks = 15

def plot_spectrogram_with_peak_spectrum(column1_data, column2_data, fs, sig_rate, save, Ver):
    # Combine the two columns of data into a single array
    data = np.array(column2_data)  # You can use column1_data or column2_data depending on which column you want to analyze

    # Perform FFT
    n = len(data)
    freqs = np.fft.fftfreq(n, 1/fs)
    fft_values = np.fft.rfft(data)

    #neu
    fft_val = 2 / len(data) * np.abs(fft_values)
    fft_values_db = 20 * np.log(fft_val)
    fft_mag = fft_values_db

    # Find peaks in the positive frequencies of the spectrum
    positive_freqs_mask = freqs >= 0
    positive_freqs = freqs[positive_freqs_mask]

    # Plot spectrogram
    plt.figure(figsize=(8, 6))

    plt.specgram(data, Fs=fs, cmap='viridis')
    plt.xlabel('Zeit [s]', fontsize=font)
    plt.ylabel('Frequenz [Hz]', fontsize=font)
    plt.xticks(fontsize=ticks)
    plt.yticks(fontsize=ticks)
    plt.title(f"Spektogramm - {fs} Hz ; Aus", fontsize=font)
    plt.colorbar(label='Signalstärke [dB]')
    if save: plt.savefig(f"analysing/auswertung/V{Ver}_SG_{fs}SR_aus.png")
    plt.show()

    plt.figure(figsize=(8, 6))
    plt.plot(freqs[:n//2], fft_mag[:n//2])  # Plot only the positive frequencies
    plt.xlabel('Frequenz (Hz)', fontsize=font)
    plt.ylabel('Amplitude', fontsize=font)
    plt.xticks(fontsize=ticks)
    plt.yticks(fontsize=ticks)
    plt.title(f'Frequenzspektrum - {fs} Hz f_a; Aus', fontsize=font)
    if save: plt.savefig(f"analysing/auswertung/V{Ver}_FS_{fs}SR_aus.png")
    plt.show()

def time_plot(sample_rate, signal_rate, save, V):
    time = []
    data = []
    with open(f'analysing/V{V}_{sample_rate}SR_aus.txt', 'r') as file:
        csv_reader = csv.reader(file)
        next(csv_reader)
```

```
    for row in csv_reader:
        val = float(row[0])
        data.append(val)

PERIOD_DURATION = 1/sample_rate
for data_points in range(len(data)):
    time.append(data_points * PERIOD_DURATION)

plt.figure(figsize=(8, 6))
plt.plot(time[0:200], data[0:200], color='blue', linestyle='-')
plt.title(f'{sample_rate} Hz Abtastfrequenz Aus', fontsize=font)
plt.xlabel('Zeit [s]', fontsize=font)
plt.ylabel('Spannung [V]', fontsize=font)
# changing the fontsize of ticks
plt.xticks(fontsize=ticks)
plt.yticks(fontsize=ticks)
plt.grid(True)
if save: plt.savefig(f"analysing/auswertung/V{V}_{sample_rate}SR_aus.png")
plt.show()

return time, data, sample_rate, save, V
```

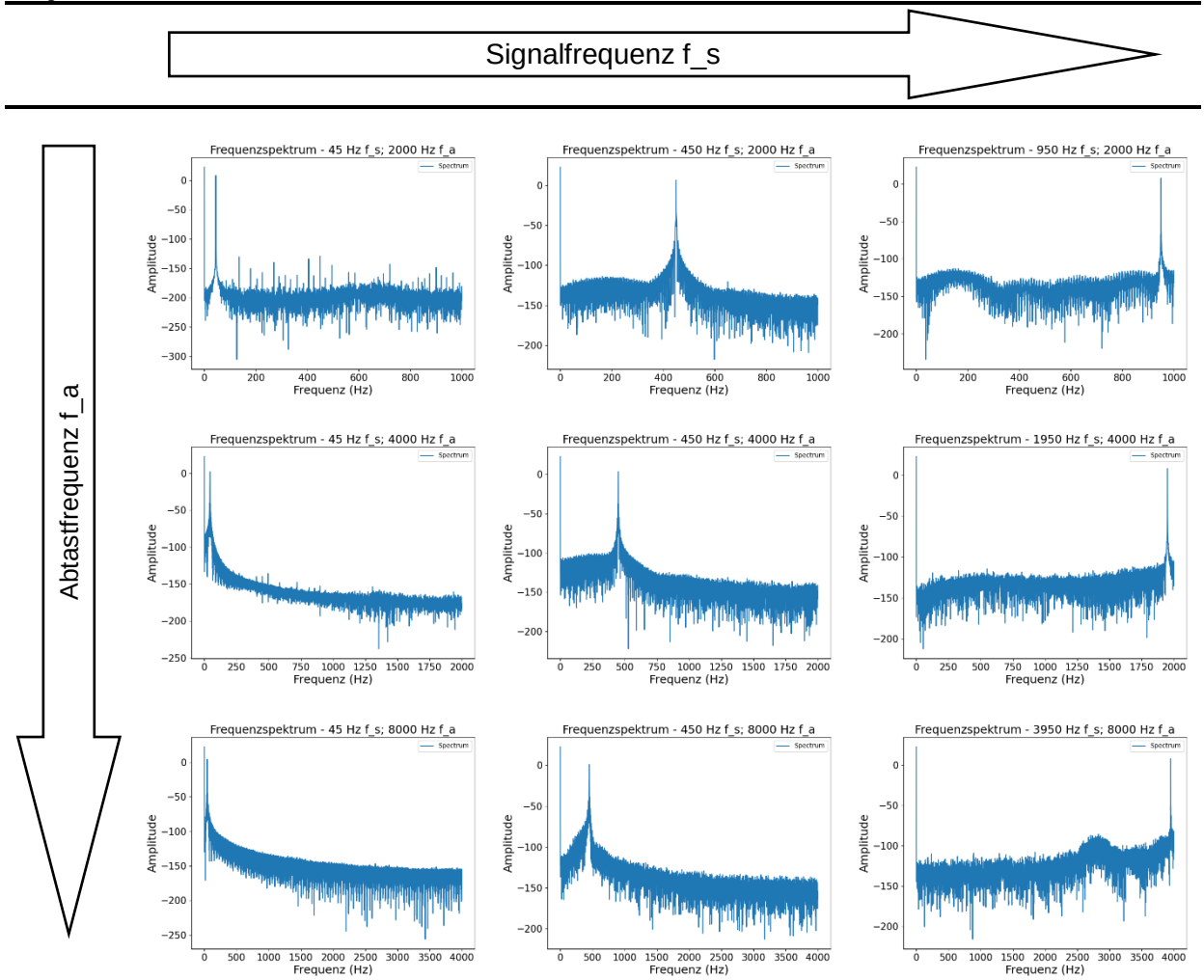
Anhang B

Tabelle 4: Übersicht der zeitlichen Darstellung unterschiedlicher Sinussignale die mit verschiedenen Frequenzen abgetastet wurden



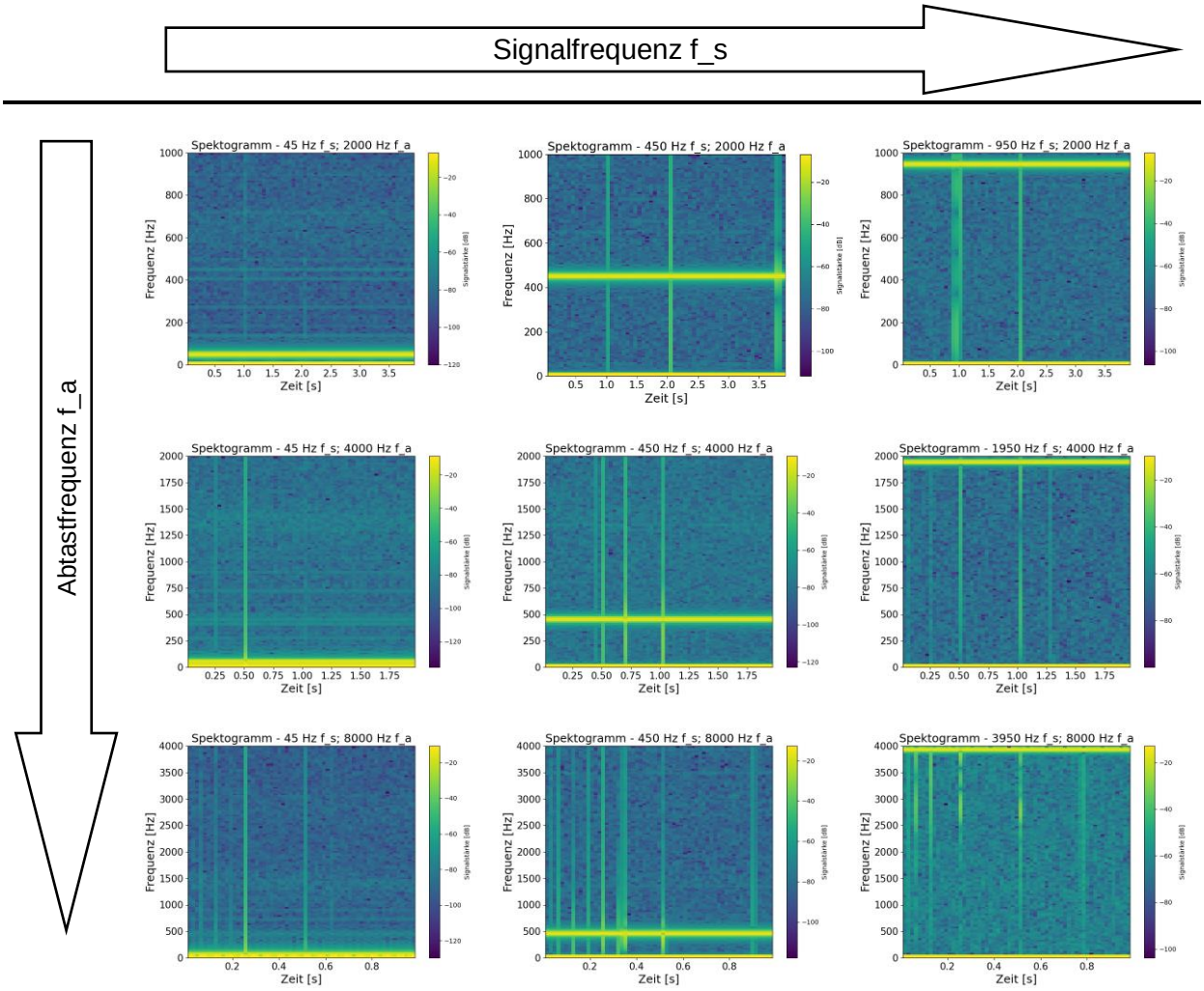
Anhang C

Tabelle 5: Übersicht der Frequenzspektren unterschiedlicher Sinussignale die mit verschiedenen Frequenzen abgetastet wurden



Anhang D

Tabelle 6: Übersicht der verschiedenen Spektrogramme verschiedener Sinussignale die mit verschiedenen Frequenzen abgetastet wurden



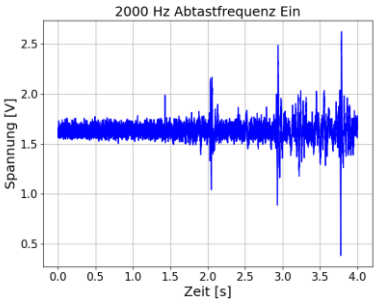
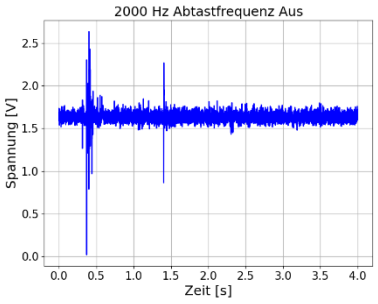
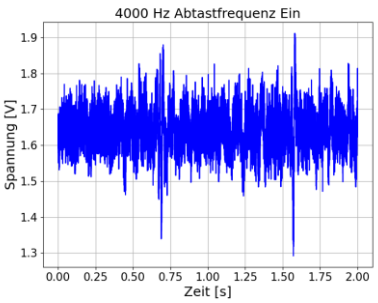
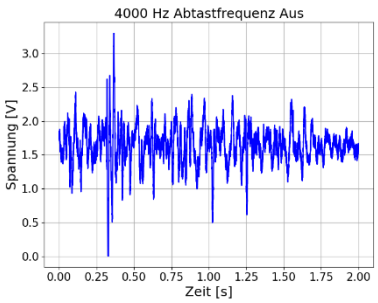
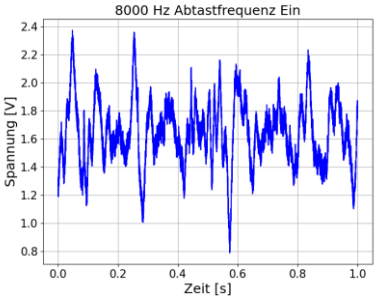
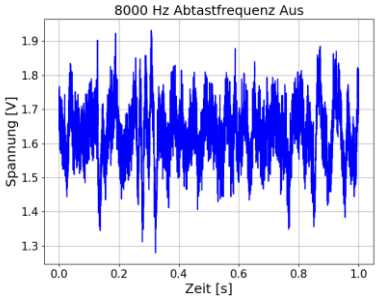
Anhang E

Code der ISR mit dem Togglen des Pins (hier als self.x wiederzufinden)

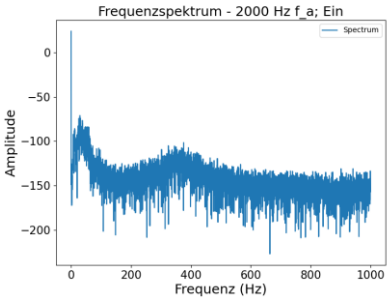
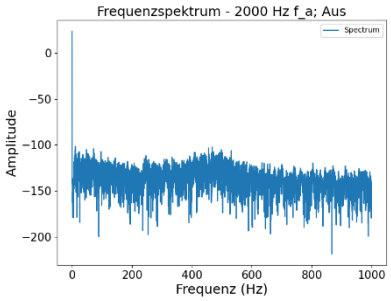
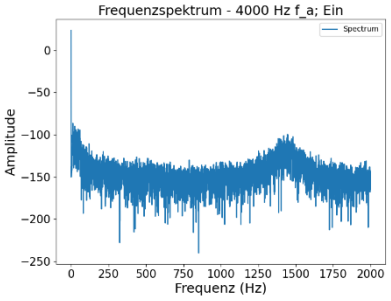
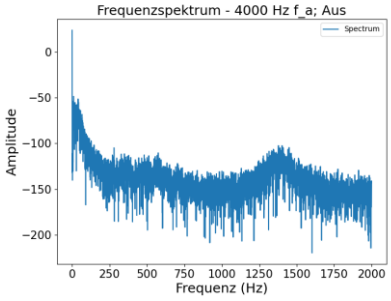
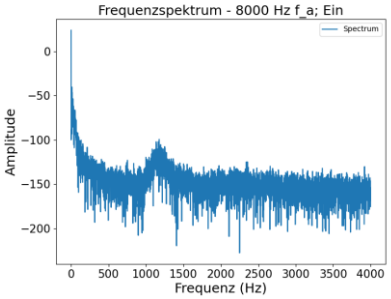
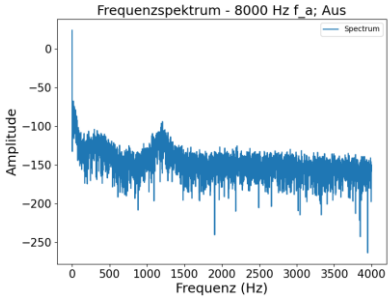
```
############################        ISRS        ############################
def __isr(self, timer):
    self.x.toggle()
    val = self.__adc.read_u16() * self.__deltaU
    self.d.append(val)
    self.count += 1
    self.x.toggle()
```

Anhang F

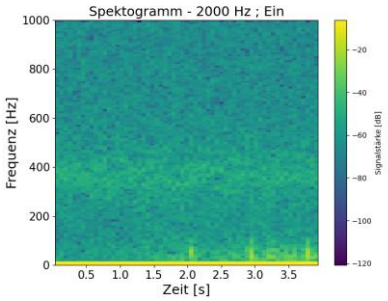
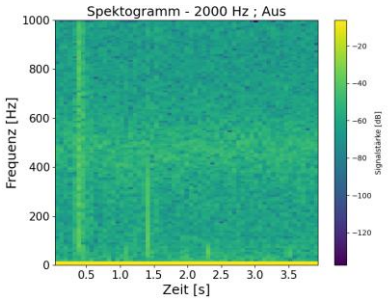
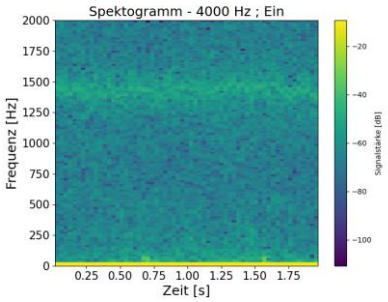
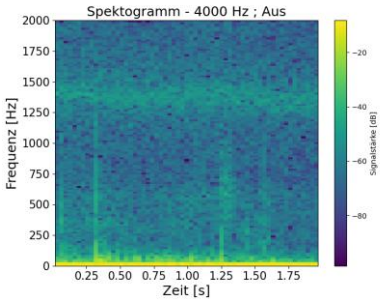
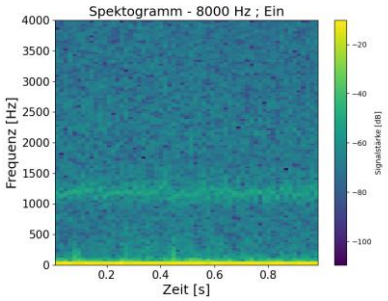
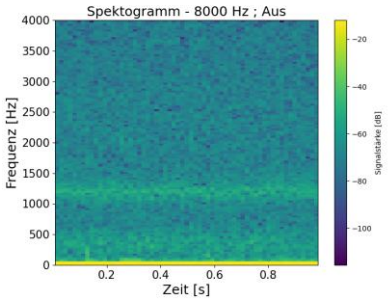
Übersicht aller gemessenen Atemzüge - Zeitbereich

Abtastfrequenz [Hz]	Einatmen	Ausatmen
2000		
4000		
8000		

Übersicht aller gemessenen Atemzüge - Frequenzspektrum

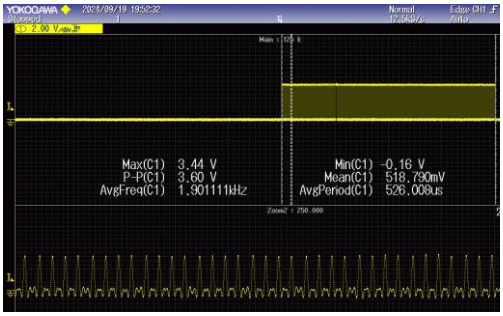
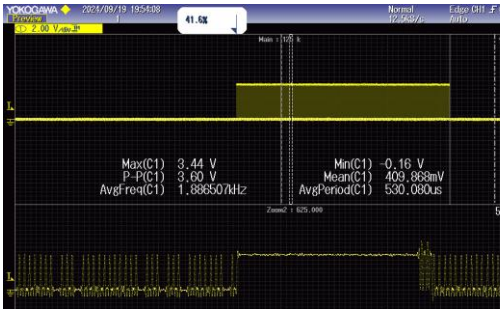
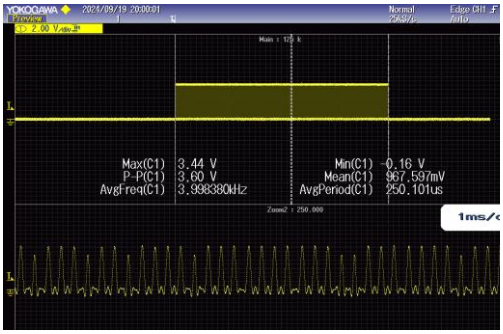
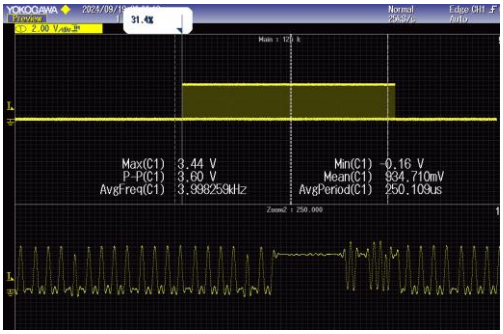
Abtastfrequenz [Hz]	Einatmen	Ausatmen
2000		
4000		
8000		

Übersicht aller gemessenen Atemzüge - Spektrogramme

Abtastfrequenz [Hz]	Einatmen	Ausatmen
2000		
4000		
8000		

Anhang G

Übersicht der mit dem Oszilloskop gemessenen Zeitdauern der ISR über das Toggeln eines Pins. Dabei wird einmal die kontinuierliche Aufzeichnung und eine Unterbrechung dargestellt

Abtastfrequenz [Hz]	kontinuierlich	Unterbrechung
2000		
4000		
8000