

Technische Hochschule Augsburg
An der Hochschule 1
D-86161 Augsburg



06.07.2025

Audio in Hardware beim PicoNut

Projektarbeit 1

Tristan Kundrat

Matrikelnummer: 2179375

Fakultät:	Informatik
Studiengang:	Technische Informatik
Semester:	SoSe2025
Kurs:	PicoNut: Ein neuer, modularer RISC-V-Prozessor
Prüfer/in:	Prof. Dr. Gundolf Kiefer
Abgabedatum:	06.07.2025

Lizenzen

© Tristan Kundrat 2025

Diese Arbeit mit dem Titel

Audio in Hardware beim PicoNut

von Tristan Kundrat steht unter folgender Lizenz:

Creative Commons Namensnennung-Nicht kommerziell-Share Alike 4.0 International



<https://creativecommons.org/licenses/by-nc-sa/4.0/deed.de>

Sämtliche, in der Arbeit beschriebene und auf dem beigelegten Datenträger vorhandene, Ergebnisse dieser Arbeit in Form von Quelltexten, Software und Konzeptentwürfen stehen unter einer

GNU General Public License Version 3

<http://www.gnu.de/documents/gpl.de.html>

Die Typst-Vorlage dieser Arbeit unter folgendem Repository:

<https://github.com/fuchs-fabian/typst-template-aio-studi-and-thesis>

wurde erstellt von © Fabian Fuchs und unterliegt der

MIT License

<https://mit-license.org/>

Erklärung zur Projektarbeit

Ich, Tristan Kundrat (Matrikelnummer 2179375), versichere hiermit, dass ich die vorliegende Arbeit mit dem Thema

Audio in Hardware beim PicoNut

selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, wobei ich alle wörtlichen und sinngemäßen Zitate als solche gekennzeichnet habe. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Emmering, den 06.07.2025.



Tristan Kundrat

Inhaltsverzeichnis

LIZENZEN	2
ERKLÄRUNG ZUR PROJEKTARBEIT	2
ABBILDUNGSVERZEICHNIS	II
1. EINLEITUNG	1
2. HAUPTTEIL	1
2.1. Anforderungen an das Audiomodul	2
2.2. Aufbau des PicoNut und des Audiomoduls	2
2.3. Generierung des Audiosignals und Registerinhalt	4
2.4. Testen des Audiomoduls	6
3. FAZIT	7
3.1. Zusammenfassung	8
3.2. Ausblick	8
LITERATUR- UND QUELLENVERZEICHNIS	9

Abbildungsverzeichnis

Abbildung 1	Aufbau des Audiomoduls	2
Abbildung 2	Aufbau des Moduls einer Audiostimme	3
Abbildung 3	Codeausschnitt aus dem Quellcode des Audiomoduls	4
Abbildung 4	Visualisierung der Berechnung der Parameter des Rechtecksignals	5
Abbildung 5	Visualisierung der Berechnung der Parameter des Sägezahnsignals	5
Abbildung 6	Visualisierung der Berechnung der Parameter des Dreieckssignals	6

1. Einleitung

Der PicoNut [1] ist ein minimaler und flexibel erweiterbarer Prozessor, der auf der RISC-V Befehlssatzarchitektur aufgebaut ist.

Um die Vielfalt an Peripheriemodulen zu erweitern, die das PicoNut-Projekt vorhält, ist das Ziel dieser Arbeit ein Audiomodul zu entwickeln, welches synthesesfähig ist. Das Audiomodul soll den Zweck eines Synthesizers erfüllen, also sollen Tonhöhe und Lautstärke konfiguriert und in Hardware ein Signal generiert werden können.

Die Arbeit ist wie folgt aufgebaut: Zunächst werden in Abschnitt 2.1 die Anforderungen an das Audiomodul vorgestellt. Anschließend wird der Aufbau des PicoNut und des Audiomoduls in Abschnitt 2.2 erläutert. In Abschnitt 2.3 wird dargestellt, wie die Generierung von den drei implementierten Audiowellen funktioniert und umgesetzt wurde. Als vorletztes kommen wir in Abschnitt 2.4 zum Testen des Moduls in Simulation, worauf eine Zusammenfassung dieser Arbeit und ein Ausblick auf mögliche Erweiterungen in Abschnitt 3 folgt.

Die Umsetzung dieser schriftlichen Ausarbeitung erfolgte mithilfe von Typst [2], einem neuen Zeichensatzsystem für wissenschaftliche Arbeiten, welches als Ersatz für andere bekannte Möglichkeiten wie LaTeX oder Word entworfen wurde.

2. Hauptteil

Soweit nicht anders angegeben, bezieht sich die Arbeit im Folgenden auf die PicoNut Dokumentation. [3]

2.1. Anforderungen an das Audiomodul

Im Exposé dieses Projekts wurde die Entwicklung und Implementierung eines einfachen Audiosynthesizers, der durch Register steuerbar ist, als Ziel gesetzt. Die Anforderungen waren:

- Generierung verschiedener Wellenformen, insbesondere:
Rechteck, Dreieck und Sägezahn
- Optional: Generierung einer Sinuswelle
- 1-Kanal-Audiomodul
- Zusammenführung mehrerer 1-Kanal-Audiomodule (Stimmen), um ein polyphones (mehrstimmiges) Audiomodul zu schaffen

2.2. Aufbau des PicoNut und des Audiomoduls

Der PicoNut-Prozessor besteht aus dem *Nucleus*, dem eigentlichen Prozessorkern, der Instruktionen ausführt, und der *Membrana* (Memory Unit), die das Interface zwischen Nucleus und *Wishbone*-Systembus darstellt.

Das Audiomodul ist am Systembus angeschlossen und kann darüber durch den Prozessor konfiguriert werden.

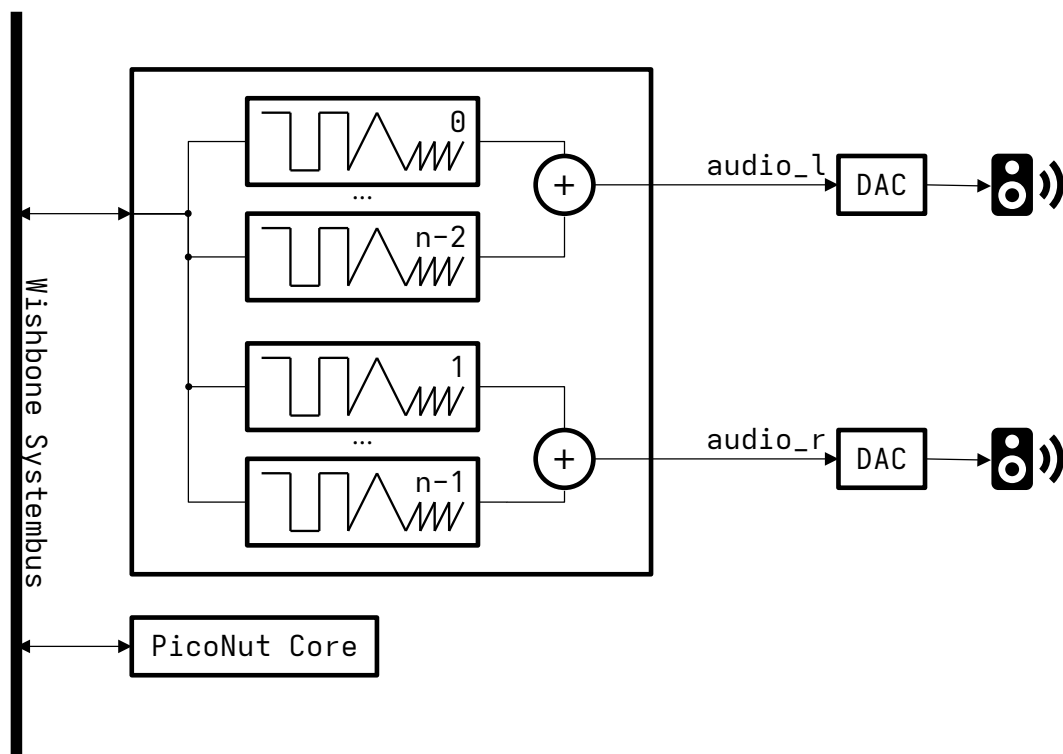


Abbildung 1: **Aufbau des Audiomoduls**

Wie in Abbildung 1 zu sehen, enthält ein Audiomodul mehrere Stimmen (Tongeneratoren), die jeweils zu einem rechten und einem linken Audiokanal zusammengeführt werden. Die Anzahl der Stimmen n insgesamt (linker und rechter Audiokanal) ist frei wählbar mit den Bedingungen $n = 2^x$ und $x \in \mathbb{N} \setminus \{0\}$.

Die Zusammenführung der Audiosignale erfolgt so: Jede Audiostimme gibt ein 16 Bit breites, vorzeichenloses Audiosignal aus. Jeweils alle geraden Stimmen von 0 bis $n-2$ werden zum linken Audiokanal `audio_l` und alle ungeraden Stimmen von 1 bis $n-1$ werden zum rechten Kanal `audio_r` mit gleicher Gewichtung (Multiplikation mit 1) zusammenaddiert.

Das Ausgangssignal des Addierers ist aufgrund der Addition breiter als die ursprünglichen 16 Bit. Um wieder ein 16 Bit breites Ausgangssignal `audio_r` und `audio_l` zu erhalten, werden die oberen 16 Bit des breiteren Ausgangssignals des Addierers als Audioausgang genutzt.

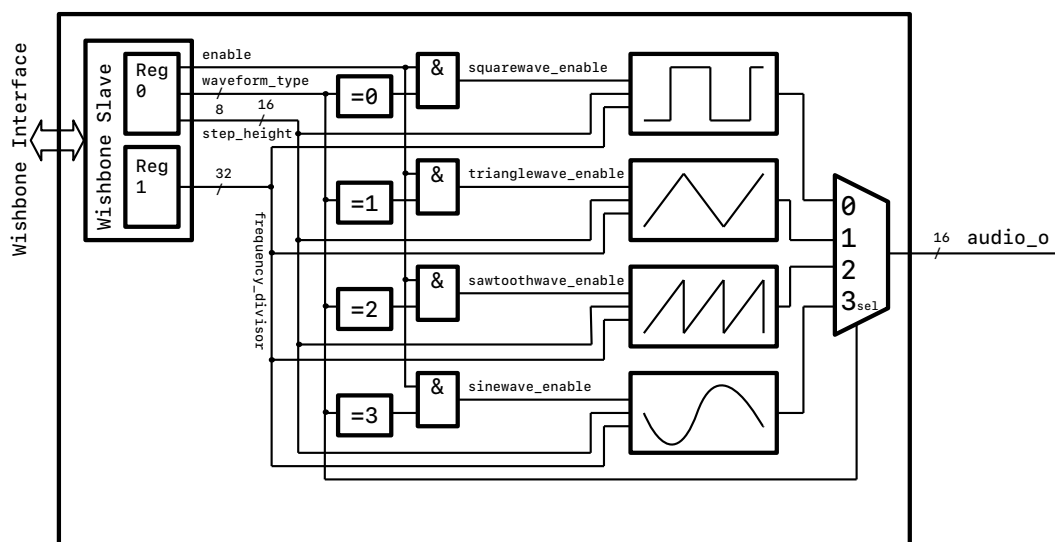


Abbildung 2: **Aufbau des Moduls einer Audiostimme**

Das Modul, in dem eine Audiostimme realisiert wird, besteht aus mehreren Komponenten (siehe Abbildung 2). Zum einen werden zwei 32 Bit breite Register angelegt, die über den Wishbone-Systembus gelesen und beschrieben werden können.

Das Register 0 enthält folgende Informationen:

- enable
- waveform_type
- step_height

Und Register 1 enthält:

- frequency_divisor

Mit waveform_type und enable wird entschieden, ob und welcher Signalgenerator aktiviert und über einen Multiplexer nach außen geleitet wird. Näheres zu den Registern und deren Bedeutung folgt in Abschnitt 2.3.

2.3. Generierung des Audiosignals und Registerinhalt

Zunächst betrachten wir uns das `enable` und den `waveform_type`.

`enable` besteht nur aus einem Bit. Für `enable = 0` ist die Audiostimme deaktiviert und alle Zählerstände werden auf 0 zurückgesetzt. Wenn `enable = 1` ist, ist die Stimme aktiv und es wird anhand aller weiteren Registerinhalte ein Audiosignal generiert.

Im Audiomodul wird ein C-Enumerable (siehe Abbildung 3) angelegt, das angibt, welche Wellenform welchem Eintrag in `waveform_type` entspricht.

```

1  typedef enum
2  {
3      WF_SQUARE = 0,
4      WF_TRIANGLE,
5      WF_SAWTOOTH,
6      WF_SINE,
7  } e_waveform_t;

```

Abbildung 3: Codeausschnitt aus dem Quellcode des Audiomoduls

`waveform_type = 0` entspricht also der Rechteckwelle, 1 der Dreieckswelle, 2 der Sägezahnwelle und 3 der Sinuswelle.

Wie auch unter Abschnitt 2.2 in Abbildung 1 zu sehen, ist ein Modul für die Sinuswelle vorgesehen. Da sich das Generieren eines Sinussignals in Hardware als zu umfangreich für diese Projektarbeit darstellt, wurde der Sinuswellengenerator angelegt, er hat jedoch keine Funktion. Diese kann jedoch zu einem späteren Zeitpunkt noch hinzugefügt werden.

Das Modul einer Audiostimme ist modular aufgebaut, sodass ohne größeren Aufwand ein neuer Typ an Wellenform hinzugefügt werden kann. Da `waveform_type` 8 Bit breit ist, können maximal $2^8 = 256$ unterschiedliche Wellenformen angesprochen werden.

Als nächstes kommen wir zum `frequency_divisor` (im Folgenden d), der für jedes Audiosignal identisch berechnet wird.

Jedem Signalgenerator liegt ein Zähler zugrunde. d ist die Zahl, durch die der Systemtakt f_{clk} geteilt wird, um die Frequenz f des gewünschten Signals zu generieren.

Die Rechenvorschrift lautet also:

$$d = \frac{f_{\text{clk}}}{f} = \frac{T}{T_{\text{clk}}} \quad (1)$$

Wobei T für die Taktperiode der Frequenz f steht. d ist eine vorzeichenlose Ganzzahl mit 32 Bit Größe.

Die `step_height` (im Folgenden s) muss für jedes der implementierten Signalgeneratoren unterschiedlich berechnet werden. s ist eine vorzeichenlose Ganzzahl mit 16 Bit Größe.

Der einfachste Fall ist die Rechteckwelle.

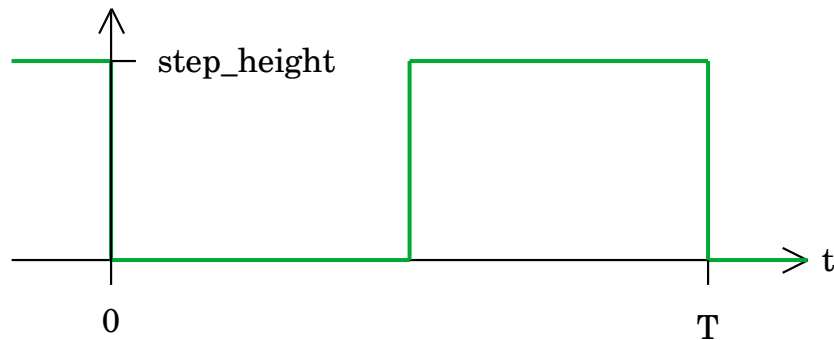


Abbildung 4: **Visualisierung der Berechnung der Parameter des Rechtecksignals**

Bei einem Rechtecksignal mit einem Aussteuergrad von 50% ist in der ersten Hälfte der Taktperiode das Ausgangssignal (audio_o, im Folgenden y) $y = 0$, in der zweiten Hälfte entspricht es der Amplitude des Signals ($y = A$).

In Abbildung 4 sehen wir, dass:

$$s_{\square} = A \quad (2)$$

Das Zählermodul, welches in jedem Signalgenerator genutzt wird, zählt von 0 bis $d-1$ und fängt dann wieder bei 0 an. Dieser Zähler hat eine interne Breite von 32 Bit und wird ohne Vorzeichen interpretiert.

Zusätzlich dazu wird bei der Sägezahn- und Dreieckswelle die Funktion genutzt, die Ausgangsamplitude jeden Takt um s zu erhöhen. Hierzu gibt es ein zweites Register mit einer Breite von 24 Bit, ebenfalls vorzeichenlos. Das Ausgangssignal audio_o sind die oberen 16 Bit dieser 24 Bit, die unteren 8 Bit werden nur zur Verbesserung der Auflösung intern genutzt (vorzeichenlose Festkommazahl mit 16 Vor- und 8 Nachkommastellen).

Somit ist die Amplitude A im Folgenden eine Zahl zwischen $0x000000$ und $0xFFFFF$.

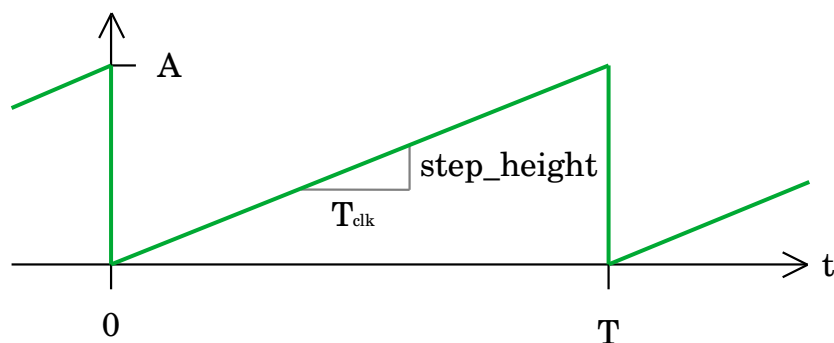


Abbildung 5: **Visualisierung der Berechnung der Parameter des Sägezahnsignals**

Wie in Abbildung 5 zu sehen, ist bei der Sägezahnwelle s von der Steigung des Signals abgeleitet.

Es muss berechnet werden, wie groß der Schritt s sein muss, um die gewünschte Frequenz und Amplitude zu erzielen.

Mit Amplitude A und Frequenz f bzw. Taktperiode T des Sägezahns ist:

$$s_s = \frac{A}{d} = \frac{A}{\left(\frac{f_{\text{clk}}}{f}\right)} = \frac{A \cdot f}{f_{\text{clk}}} = \frac{A}{\left(\frac{T}{T_{\text{clk}}}\right)} = \frac{A \cdot T_{\text{clk}}}{T} \quad (3)$$

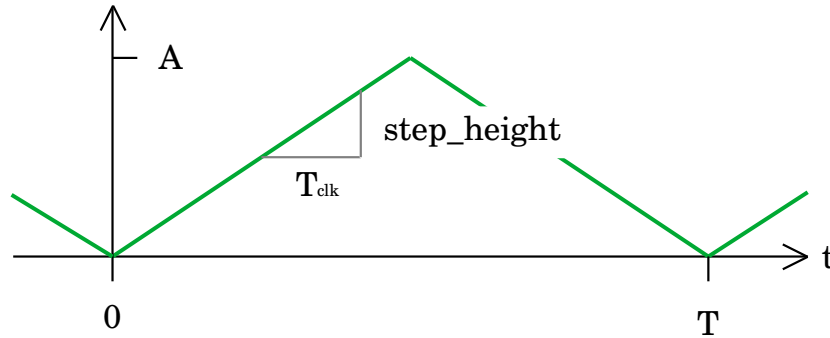


Abbildung 6: **Visualisierung der Berechnung der Parameter des Dreieckssignals**

In Abbildung 6 kann man erkennen, dass s_{Δ} hier der gleichen Logik wie s_s des Sägezahns entspricht.

Die Steigung wird nur verdoppelt, daher folgt:

$$s_{\Delta} = 2 \cdot s_s = \frac{2A}{d} \quad (4)$$

2.4. Testen des Audiomoduls

Um das erstellte Audiomodul zu testen, wurde zunächst eine *Testbench* erstellt, die das komplette Audiomodul instanziiert, wobei ein Aufbau mit insgesamt acht Stimmen gewählt wurde.

Die Testbench agiert hierbei als *Wishbone-Master*, um das Audiomodul ohne den PicoNut Core simulieren zu können, da dies sehr rechenintensiv ist. Über den Systembus werden die einzelnen Audiostreams unterschiedlich konfiguriert (also deren Register beschrieben), die schließlich das Signal generieren.

Nun kann die Testbench ausgeführt und ein *Tracefile* generiert werden. Das Tracefile enthält alle Signale, die innerhalb des Audiomoduls vorliegen und zur Aufzeichnung markiert wurden. Mithilfe eines geeigneten Programms können die Signalverläufe aus dem Tracefile visuell dargestellt werden.

Da es sehr komplex werden würde, die Ausgabe des Audiomoduls direkt in der Testbench mittels vergleichenden assert-Anweisungen zu testen, wurde ein anderer Aufbau gewählt.

Es wurde ein kurzes Programm `compare_vcd.py` erstellt, welches die SHA256 Prüfsumme des Tracefile berechnet, nachdem irrelevante Metadaten (bspw. Erstellungsdatum) abgeschnitten wurden. Diese Prüfsumme wird mit einer bekannten Prüfsumme verglichen. Stimmen diese überein, ist die Ausgabe des Audiomoduls korrekt, ansonsten nicht.

Um die Prüfsumme, gegen die geprüft wird, zu erstellen, muss zunächst das Tracefile generiert werden. Anschließend muss das Tracefile händisch (über visuelle Darstellung) betrachtet werden, ob die Ausgaben der Erwartung entsprechen. Ist das der Fall, kann das erstellte Skript mit dem Parameter `update` aufgerufen werden. Wenn wir uns im Verzeichnis des Programms befinden, lautet der Aufruf:

```
1 $ ./compare_vcd.py update
```

Hierbei berechnet das Skript wie zuvor die Prüfsumme des Tracefile und speichert diese ab, sodass dies die neue Prüfsumme wird, mit der in Zukunft verglichen wird.

Durch diesen Test können wir Regressionen erkennen. Wenn also durch eine spätere Änderung am PicoNut-Projekt erfolgt, die Auswirkungen auf das Audiomodul hat, werden diese durch den Test erkannt und mögliche Fehler können behoben werden.

Zusätzlich zum Simulationstest wurde das Audiomodul auch in Hardware auf dem ULX3S getestet, ein vollständig quelloffenes und preisgünstiges FPGA-Board, welches seit 2016 in Zagreb in Kroatien entwickelt wird. [4]

Für diesen Hardwaretest wurde ein Programm entwickelt, welches über Speicherzugriffe auf das Audiomodul über den Wishbone-Systembus die gewünschten Register konfiguriert, um eine Melodie abzuspielen.

3. Fazit

3.1. Zusammenfassung

In dieser Projektarbeit wurde ein synthetisierbares Audiomodul für den PicoNut in der Programmiersprache C++ entwickelt. Das Modul enthält mehrere Stimmen, deren Anzahl konfigurierbar ist. Die Stimmen können über den Wishbone-Systembus konfiguriert werden, wodurch die gewünschten Signale generiert und an den beiden Audioausgängen ausgegeben werden. Es ist möglich ein Rechteck-, Dreieck- und Sägezahnsignal zu generieren. Zudem läuft das Modul sowohl in Simulation, als auch auf echter FPGA-Hardware.

3.2. Ausblick

Die Sinuswelle wurde aus Gründen des Zeitumfangs ausgelassen. Das Modul ist jedoch so aufgebaut, dass diese Sinuswelle (oder auch andere Wellenformen) noch nachträglich ohne größeren Programmieraufwand hinzugefügt werden kann.

Auch denkbar ist die Erweiterung durch einen Audiomixer, der statt einer einfachen Addition der einzelnen Stimmen zu einem Audioausgang mit gleicher Gewichtung, auch unterschiedliche Gewichtungen unterstützt. Diese Gewichtungen könnten dann ebenfalls über den Systembus konfiguriert werden. So könnte beispielsweise einfach ein „Raumklang“ erzeugt werden, der weniger Stimmen als mit dem jetzigen Modul benötigt.

Literatur- und Quellenverzeichnis

- [1] „PicoNut - Forschungsgruppe Effiziente Eingebettete Systeme“. Zugegriffen: 19. Juni 2025. [Online]. Verfügbar unter: <https://ees.tha.de/piconut/index.html>
- [2] „Typst Documentation“. Zugegriffen: 1. Juli 2025. [Online]. Verfügbar unter: <https://typst.app/docs/>
- [3] G. Kiefer, M. Schäferling, und J. Hofmann, „PicoNut Documentation (CC-BY 4.0, <https://creativecommons.org/licenses/by/4.0/>)“. Zugegriffen: 19. Juni 2025. [Online]. Verfügbar unter: <https://ees.tha.de/piconut/manual/>
- [4] „ULX3S Webpage“. Zugegriffen: 2. Juli 2025. [Online]. Verfügbar unter: <https://radiona.org/ulx3s/>