Kommunikationstechnische Anwendungen für labAlive Web und App

Johannes Krüger (B. Eng.)

Prüfer: Prof. Dr. -Ing. Erwin Riederer

Masterarbeit

Eingereicht im August 2023

ETTI 5 - Institut für Funkkommunikation
Universität der Bundeswehr München

ERKLÄRUNG

Hiermit erkläre ich i. S. des § 35 Abs. 7 RaPO,

- dass ich die vorliegende Bachelorarbeit selbständig verfasst habe,
- dass ich diese Arbeit noch nicht anderweitig für Prüfungszwecke vorgelegt habe,
- dass ich keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe,
- dass ich wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

_____ _____

Ort, Datum Unterschrift

Inhaltsverzeichnis

1. Einleitung

1.1 Motivation

Wenn es einen Wirtschaftszweig gibt, der unser Leben in den letzten Jahrzehnten maßgeblich verändert hat, so ist es sicherlich die Telekommunikation. Nach den Ergebnissen des „Digital 2023: Global Overview Report“ von „DataReportal“ [1] verbringt der durchschnittliche Europäer etwa 6 Stunden vor dem Bildschirm. Ganz gleich ob es sich hierbei um das eigene Mobiltelefon, dem PC oder dem Fernseher handelt. Und mit Blick über den Atlantik steigt diese Zahl sogar auf knappe 7 Stunden für den durchschnittlichen US-Bürger, oder gar ganze 9.5 Stunden für den durchschnittlichen Brasilianer. Telekommunikation ist aus dem heutigen Leben wie wir es kennen schlichtweg nicht mehr wegzudenken. Viel zu sehr ist es mit anteiligen knapp 30%-50% unserer Wachzeiten in unserem Alltag verankert. Das Mobilfunkgerät wurde zum treuen und unersetzlichen Begleiter durch den Tag. So sehr, dass wir einige Dinge für selbstverständlich halten, darunter die Tatsache, dass vor einem Jahrzehnt viele dieser Dinge nur schwer möglich oder zumindest viel teurer waren. Es handelt sich in der Tat um eine Entwicklung, die durch das Kundenverhalten bestimmt wird, die aber auch einen großen Einfluss darauf hat, wie wir Daten konsumieren, wie wir Einkäufe tätigen und wie wir, nach der Statistik von „DataReportal“, letztendlich unsere Zeit verbringen (siehe Abb. 1 Average Screen Time by Country).

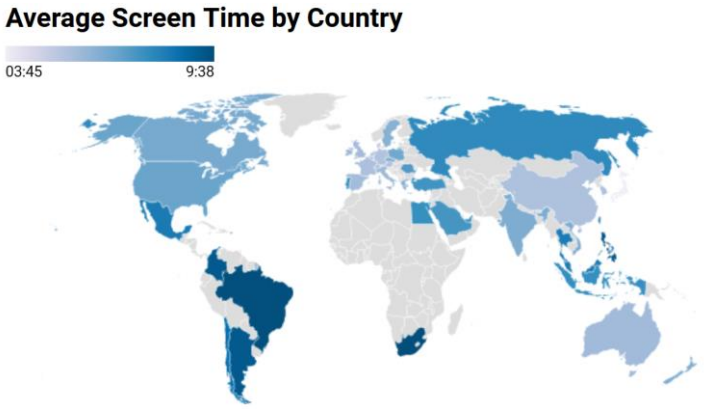


Abbildung 1 Average Screen Time by Country
(https://www.comparitech.com/tv-streaming/screen-time-statistics/)

Doch hinter all der Selbstverständlichkeit in der alltäglichen Nutzung stecken komplexe Hintergründe, wie das große Kapitel der Signalverarbeitung und die Vielzahl an Kommunikationsstandards, die diese Masse an täglichen Datentransfers überhaupt erst unterstützbar und möglich machen.

Diese hohe Anzahl, zusammen mit dessen positiven Trend, erfordern immer leistungsfähigere Techniken und es braucht auch in Zukunft Innovationen, um die Vernetzung unserer Welt leistbar zu halten. Und genau hierin ergründet sich die Anwendung „labAlive“ . Ein für die Lehre entwickeltes Tool, dass Interessierten die Themen der Signalverarbeitung und Telekommunikation näherbringt. Und dem Nutzer, ganz ohne die Notwendigkeit an teure telekommunikationstechnische Hardware, den Weg hin zur kommunikationstechnischen Forschung ebnet.

1.2 LabAlive

Die Plattform labAlive wurde von Prof. Dr. Erwin Riederer entwickelt, ist in der Fakultät ETTI (=Elektrotechnik und Technische Informatik) der Universität der Bundeswehr München eingebettet und dient seit jeher der Vermittlung telekommunikationstechnischen Wissens, anhand Applikationsunterstützter Experimente.

Diese decken dabei eine Spanne ab von der fundamentalen Spektrumsanalyse mittels der „FFT: Fourier Transformation“, über die Vermittlung unterschiedlicher De-/Modulationsarten, sowohl aus der analogen wie auch aus der digitalen Signalwelt, bis hin zum experimentellen Entdecken der Ausbreitungseigenschaften und Phänomenen von Signalen im freien Raum.

Diese in Java implementierten Experimente bilden hierbei zugleich das Herzstück von labAlive und wurden mit englischsprachigen Tutorials und Anleitungen versehen, um die Plattform nicht nur für all jene Studierende der Universität der Bundeswehr München, sondern auch für das internationale Publikum zugänglich zu machen.

Um den Nutzern weitere Anpassungsmöglichkeiten zu bieten, wurde das Tool "myLabAlive" entwickelt. Mit diesem können individuellen Applikationen nach eigenen Vorstellungen und Lernbedürfnissen erstellt werden. In diesem anwenderorientierten Nutzerbereich können angemeldete Benutzer ihre eigenen Experimente erstellen, verwalten und anpassen. Die Erzeugung erfolgt dabei durch die Programmierung innerhalb eines Textfeldes (genannt Text2App). Dort kann mit idealisierten Bauteilen und deren Parametern ganze Schaltungen generiert werden und mit Programmierbefehlen und System- bzw. Applikationseinstellungen experimentiert werden.

Jenes Tool „myLabAlive“ ist hierbei zugleich Hauptgegenstand dieser Masterarbeit und die damit verbundene Aufgabenstellung wird im Folgenden näher beleuchtet.

1.3 Aufgabenstellung

Diese Masterarbeit ist das Ergebnis einer kollaborativen Zusammenarbeit zwischen den diesjährigen Bachelorstudierenden unter der Betreuung von Prof. Dr.-Ing. Erwin Riederer und den beiden Masterstudierenden, einschließlich dieser Arbeit.

Hierbei galt es gemeinschaftlich die Funktionalität von „myLabAlive“ zu überarbeiten und das Tool als solches in die nächste Entwicklungsstufe zu heben. Die jeweilige Aufgabenverteilung orientierte sich hierbei an den unterschiedlichen Studienschwerpunkten und Fachkenntnissen aller Beteiligten. So implementierten und verbesserten die Studierenden mit IT/Cyber-Studienschwerpunkt beispielsweise die Text2App-Funktionalität von „myLabAlive“. Also die Programmierumgebung über die der Anwender später eine eigene App gestalten wird. Hingegen sich diese Masterarbeit speziell auf den kommunikationstechnischen Bereich konzentriert (CT-Schwerpunkt).

So umfasste die Aufgabenstellung die Betreuung und Unterstützung der anderen Projekte im Hinblick auf kommunikationstechnische Fragestellungen. Des Weiteren wurden Überarbeitungen im Tutorial und der Dokumentation von „myLabAlive" vorgenommen. Ziel war es, den Nutzern einen noch zugänglicheren Einstieg als bisher in das Tool zu ermöglichen, um die Benutzererfahrung insgesamt zu optimieren.

Ein weiterer wesentlicher Aspekt dieser Masterarbeit bestand darin, relevante Showcase-Experimente zu konzipieren und zu integrieren. Diese Experimente sollten nicht nur die vielfältigen Funktionen und Möglichkeiten von „myLabAlive" präsentieren, sondern auch den Anwendern einen praktischen und anschaulichen Einblick bieten. Hierbei wurde großer Wert auf die Generierung und Auswahl von sinnvollen Experimenten und Erläuterungen gelegt, um die Nutzer optimal bei der Anwendung des Tools zu unterstützen und zum Experimentieren zu animieren.

Parallel zu diesen Erweiterungen und Verbesserungen wurden ausgewählte und bereits vordefinierte Projekte aus dem Experimenten-Pool von „labAlive" herausgegriffen und überarbeitet. Ziel war es, diese Projekte an die neuesten Entwicklungen anzupassen und auf den aktuellsten Stand zu bringen.

Die Durchführung aller Arbeiten erfolgte unabhängig voneinander, jedoch wurden sie durch regelmäßige wöchentliche Meetings koordiniert und aufeinander abgestimmt. Diese Meetings dienten nicht nur dazu, den Fortschritt der einzelnen Arbeiten zu überprüfen und voranzutreiben, sondern auch dazu, wertvolles Feedback auszutauschen. Insgesamt bildet die Weiterentwicklung von „myLabAlive" den zentralen Kern dieser Masterarbeit, wobei jedoch auch Arbeitszeit für die Verbesserung des Experimentes der "Analog demodulation“ aus dem Pool der LabAlive-Experimente investiert wurde.

1.4 Aufbau und Gliederung der Masterarbeit

Das erste Kapitel widmete sich der Präsentation des Online-Labors „labAlive“ und der Beschreibung der eigenen Rahmenbedingungen und Aufgabenstellung dieser Masterarbeit. Die nachfolgenden Kapitel 3 und 4 setzen sich mit der verwendeten Software zur Umsetzung auseinander und geben einen Überblick über die wichtigsten kommunikationstechnischen Grundlagen der verwendeten Systeme in den Experimenten. Diese dienen als theoretische Einführung, um dann in Kapitel 4 zur Durchführung überzugehen. Abschließend wird in Kapitel 5 und 6 ein Fazit der Arbeit gezogen und ein Ausblick auf mögliche weiterführende Arbeiten gegeben. Die Kapitel 7 und 8 enthalten den Anhang sowie das Literaturverzeichnis, das die in dieser Arbeit verwendeten Quellen und Verweise umfasst.

2. Programme

2.1 Eclipse

Die von IBM im Jahr 2001 als „VisualAge“-Nachfolger konzipierte Java-Entwicklungsumgebung gilt heute als eine der etabliertesten integrierte Entwicklungsumgebung (IDE), die für die Softwareentwicklung verwendet wird. Eclipse wurde ursprünglich als Entwicklungsumgebung für Java konzipiert, hat sich jedoch im Laufe der Zeit weiterentwickelt und unterstützt heute eine breite Palette an Programmiersprachen. Neben Java bietet Eclipse unteranderem Unterstützung für Sprachen wie C/C++, Python, PHP, Ruby und JavaScript. Entwickler können Eclipse somit als vielseitige Plattform nutzen, um Softwareprojekte flexibel in ihren bevorzugten Sprachen zu erstellen und zu verwalten.

Die IDE ist mit einer Vielzahl von Funktionen ausgestattet, darunter ein Code-Editor, Debugger, Compiler, automatische Codevervollständigung, Projektverwaltung und Versionskontrolle. Es ermöglicht Entwicklern, effizient und produktiv Softwareprojekte zu generieren, zu bearbeiten und letztlich unterstützend mit dem Debugger auf Lauffähigkeit zu testen.

Darüber hinaus ist Eclipse eine Open-Source-Software, die dank einer großen Community von Entwicklern zu einer flexiblen und anpassbaren Entwicklungsumgebung wurde. So stehen der erweiterbaren Plattform Erweiterungen und Plug-ins für spezifische Anwendungsbereiche und Anforderungen zur Verfügung, die die Funktionalität der IDE anpassen oder erweitern.[2]

2.2 Git

Eine solche Plug-in Erweiterung stellt „Git“ dar.

Git ist ein bewährtes Versionskontrollsystem, das die Verfolgung von Änderungen in Codeprojekten ermöglicht.

Durch die Integration von Git in Eclipse lassen sich Projekte direkt in der IDE verwalten, Versionen verfolgen, Branches erstellen, Änderungen überprüfen und diese mit anderen Entwicklern abstimmen. Dies erleichtert die effiziente Verwaltung des Quellcodes und ermöglicht es den Entwicklern, Änderungen an ihrem Code zu verfolgen und bei Bedarf wiederherzustellen.

Mit Git für Eclipse können Entwickler auch auf gängige Git-Funktionen wie Commits, Pushes, Pulls, Merges und das Überprüfen von Änderungen zugreifen, ohne die IDE verlassen zu müssen. Dadurch wird der Entwicklungsprozess nahtlos in Eclipse integriert und ermöglicht eine effiziente und kollaborative Entwicklungsumgebung, in der auf die leistungsstarken Tools sowohl von Eclipse als auch von Git zurückgegriffen werden kann.[3]

2.3 OBS Studio

OBS Studio steht für „Open Broadcaster Software Studio“ und ist eine Open-Source-Software, die zur Aufnahme, Bearbeitung und Übertragung von Videoinhalten verwendet wird. Bekannt ist die Software vor allem durch sogenannten „Content Creatorn“ aus der Streaming- und Gaming-Szene, da diese einen großen Nutzen aus dessen breiten und kostenlosen Funktionspalette ziehen. So bietet OBS Studio unteranderem die Möglichkeit der Video- und Audioaufnahme, bietet Echtzeit-Editing mit Filteranpassungen sowie personalisierter Szeneneinstellung für Streaming-Plattformen, wie das Hinzufügen von Webcams, Bilder und Texte zur bestehenden Bildschirmaufnahme.

Neben dem Livestreaming bietet OBS Studio jedoch auch die Möglichkeit, Inhalte lokal aufzunehmen. Benutzer können ihre Streams oder Videos in verschiedenen Formaten, Auflösungen und Bitraten aufzeichnen.

Insgesamt bietet OBS Studio eine leistungsstarke und vielseitige Plattform für Content Creator, um hochwertige Videoinhalte zu erstellen und zu übertragen. Die Software wird kontinuierlich weiterentwickelt, kann durch Plug-ins erweitert werden und steht plattformübergreifend sowohl dem Betriebssystem Windows als auch Mac und Linux Nutzern frei zur Verfügung.[4]

2.4 DaVinci Resolve

DaVinci Resolve gehört neben Adobe Premiere Pro und Final Cut Pro zu den geläufigsten professionellen Videobearbeitungssoftwares und wurde 2004 durch das US-amerikanische Unternehmen „DaVinci Systems“ gegründet und 2009 von der australischen Firma „Blackmagic Design“ übernommen und kontinuierlich weiterentwickelt.

Wodurch DaVince Resolve seine Marktkonkurrenten jedoch hinter sich lässt, ist durch seine zusätzliche fortschrittliche und durch KI unterstützte Farbkorrektur- und Farbgradierungsfunktion. So vereint es neben den marktüblichen Videobearbeitungsfunktionen wie Schneiden, Trimmen, Zusammenfügen, visueller Effekteinsatz, Einfügen von Videoübergängen und Organisieren von Videoclips auch eine äußerst präzise Farbanpassung für Videos.

Die Software unterstützt auch die Bearbeitung und Mischung von Audiospuren, um eine klare und professionelle Klangqualität zu erzielen. Hierdurch und in Verbindung mit seiner prominenten Farbkorrektur- und Farbgradierungsfunktion bietet es eine umfassende Lösung für die gesamte Postproduktion, wodurch es vor allem in der Film- und Fernsehindustrie zu einem beliebten und verbreiteten Tool wurde.

Ein weiterer Vorteil von DaVinci Resolve ist dessen Fähigkeit zur effizienzsteigernden Projektteilung, so können mehrere Anwender gleichzeitig an einem Projekt arbeiten.

Letztlich bietet Blackmagic Design ihre Software unter zwei Varianten an. Einer bereits umfangreichen und kostenlosen Variante und einer kostenpflichtigen, die dann jedoch als „DaVinci Resolve Studios“ vermarktet wird.

Aufgrund der genannten Alleinstellungsmerkmale bei zugleich kostenfreier Abwandlung des vollwertigen Programms, wurde es zu einer beliebten Wahl für Filmemacher, Videobearbeiter und Content-Ersteller und konnte sich obendrein zu eine der führenden Softwarelösungen in der Branche etablieren. [5]

3. Theoretische Grundlage

3.1 (De-)Modulationsarten

Nachrichten, seien sie text-, sprach-, bild, oder videobasiert, müssen für eine sinnvolle Übertragung an den physikalischen Eigenschaften des Übertragungsweges angepasst werden - dieser Prozess wird Modulation genannt. Durch diese wird die Nachricht aus ihrer ursprünglichen Form (=Quellensignal) in ein Sendesignal gewandelt, um es samt seinen charakterisierenden Größen der Amplitude, Frequenz und Phase an die Gegebenheiten anzupassen.

Erst durch diese Aufbereitung der Informationen in eine andere Signalform, lassen sich die zu erzielende Kriterien wie **vorgegebene Entfernungen**, **Qualitätsansprüche** wie Störabstand beziehungsweise Störschutzsicherheit und die **ökonomischen Rahmenbedingungen** wie die Kanalkapazität und die spezifischen Eigenschaften des Übertragungskanales (zeit- und frequenzabhängige Kanaldämpfungen) berücksichtigen und verwirklichen.

Hierfür haben sich in der Geschichte der Telekommunikation unterschiedliche Modulationsarten entwickelt, die sich in die Gruppen der „analogen“ und „digitalen“ Modulationsverfahren unterteilen lassen. In erster und allgemeiner Unterscheidung lassen sich diese Obergruppen so differenzieren, dass sämtliche analogen Modulationsverfahren die Zielsetzung verfolgen, den Zeit- und Werteverlauf eines Nachrichtensignals realitätsgetreu zu erhalten und Störungen des Kanals zu unterbinden, während die Modulationsverfahren der digitalen Gruppe durch ihre Abstraktion des Zeit- und Wertebereichs zusätzliche Codierung einbinden, um dadurch mögliche Übertragungsfehler zu berichtigen. [6]

Im Folgenden werden ausgewählte und die für den Tätigkeitsbereich der Masterarbeit relevante (De-)Modulationsarten hergeleitet und deren Vorteile und Einsortierung in die Obergruppen aufgezeigt.

Es sei jedoch betont, dass die Modulations- und Demodulationsverfahren lediglich in ihrer idealisierten Form (störungsfrei) hergeleitet werden, so wie sie auch im Tätigkeitsbereich dieser Masterarbeit vorkamen.

3.1.1 Amplitudenmodulation

Die Amplitudenmodulation auch kurz „AM“ genannt, zählt in der Kommunikationstechnik mit zu den grundlegendsten analogen Modulationsarten und wird zur Übertragung von Informationen, insbesondere von Audiosignalen genutzt. Das Modulationsverfahren der Amplitudenmodulation lässt sich hierbei noch in vier Unterverfahren aufteilen, das der „Zweiseitenband-Amplitudenmodulation mit Träger“, der „Zweiseitenband-Amplitudenmodulation mit unterdrücktem Träger“, der „Einseitenband-Amplitudenmodulation“ und der „Restseitenbandmodualtion“, wobei letzteres sich als Sonderfall in die digitale Obergruppe einsortiert und hier nur aus Gründen der Vollständigkeit aufgelistet ist.

Aufgrund des konventionelleren Ansatzes wird im Folgenden das Amplitudenmodulationsverfahren angelehnt am Unterverfahrens „Zweiseitenband-Amplitudenmodulation mit unterdrücktem Träger“ hergeleitet.

Bei der Amplitudenmodulation wird das Nachrichtensignal auf die Amplitude des Trägersignals aufgeprägt. Dies wird durch eine Multiplikation zwischen Quellensignal und hochfrequent oszillierenden Trägersignals erreicht und erbringt ein sinusförmiges Signal, dessen Amplitude von dem Nachrichtensignal abhängig ist. Die Frequenz des Signals ist dabei konstant. Abbildung 2 zeigt den typischen Aufbau eines AM-Modulators und dessen vorhergehendes Quellsignales (m(t)) und finalen Sendesignals (t(t)) in der Oszilloskopdarstellung.[7]

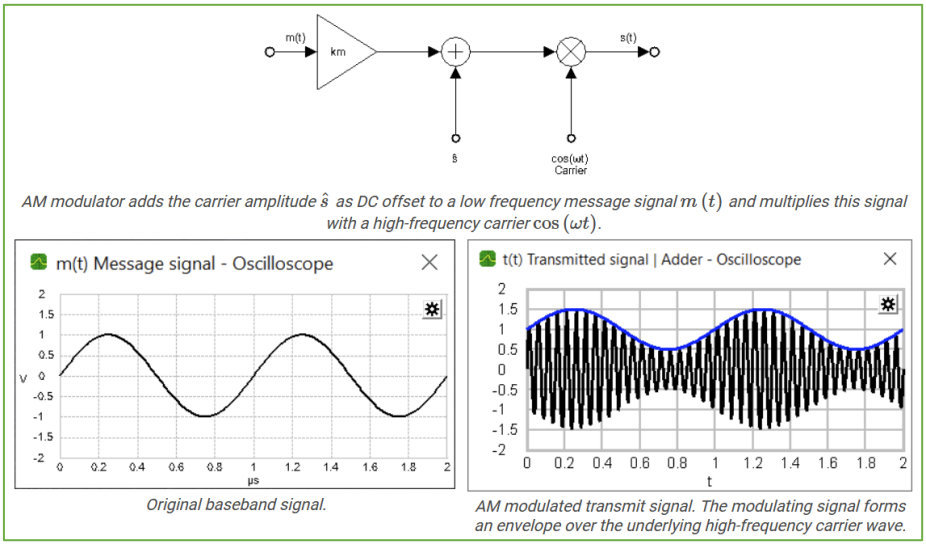


Abbildung 2 AM Modulator - labAlive Tutorial (https://www.etti.unibw.de/labalive/experiment/am/)

Mathematisch lässt sich die AM-Modulation als eine Funktion der Multiplikation aus $\hat{s}$-Trägeramplitude, $m(t)$-Quellensignal und k_m-Empfindlichkeitsgröße des AM-Modulators mit dem hochfrequenten Trägersignal "$cos(\omega t)$" definieren:

$$s(t) = [\hat{s} + m(t) \times k_m] \times cos(\omega t)$$

Die hier aufgezeigte Struktur ist zugleich der größten Vorteile dieses Modulationsverfahrens, da sich diese technisch simple realisieren lässt.

Dem gegenüber steht jedoch die damit einhergehende hohe Störanfälligkeit als Amplitudenschwankung durch beispielsweises Fading. Zusätzlich gilt die benötigte Bandbreite als teuer für die Frequenzbandaufteilung, da, wie in Abbildung 3 gezeigt, bei der „Zweiseitenband-Amplitudenmodulation“ sowohl das obere als auch das untere Seitenband (USB= Upper Side Band, LSB= Lower Side Band) das gleiche Spektrum des Quellensignals tragen (Banderweiterungsfaktor J=2) und damit eine redundante Signalkomponente eingebaut ist. Ein besonderer Nachteil des „Zweiseitenband-Amplitudenmodulation mit Träger“ ist es außerdem, dass ein nicht unwesentlicher Anteil der Sendeleistung für den nachrichtenlosen Träger einkalkuliert werden muss (ca. 80%), was zu einem geringen Modulationsgewinn von beispielsweise M=0,14 führt.

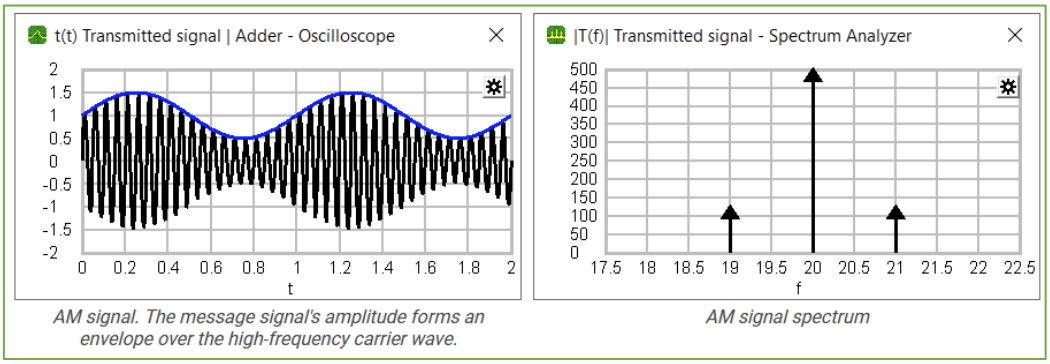


Abbildung 3 AM Modulator - labAlive Experiment (https://www.etti.unibw.de/labalive/experiment/am/)

Dies führt dazu, dass dieses Modulationsverfahren heutzutage hauptsächlich nur noch im Kurz- und Mittelwellenrundfunk eingesetzt wird und dessen Klangqualität im Vergleich zum UKW-FM-Rundfunk dürftig ausfällt.

Des Weiteren kann über die maximale und minimale Amplitude des finalen Sendesignals die Größe des Modulationsgrades „m“ gebildet werden:

$$m = \frac{s_{max} - s_{min}}{s_{max} + s_{min}} = \frac{s_{max} - s_{min}}{2\hat{s}}$$

$$\text{bzw.} \quad m = \frac{m_{max} \times k_m}{\hat{s}}$$

Dieser dient als Index dafür, wie stark das modulierte Nutzsignal die Amplitude des Trägersignals beeinflusst hat, muss für eine inkohärent Demodulation zwischen 0 und 1 liegen und lässt sich zur Veranschaulichung auch grafisch in Form eines sogenannten Modulationstrapezes darstellen, siehe Abbildung 4.

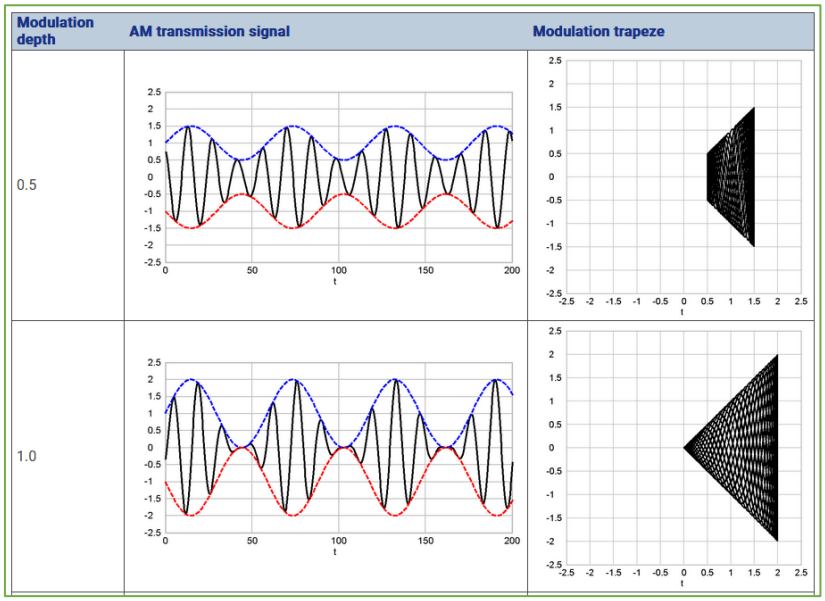


Abbildung 4 AM Modulator - labAlive Illustration (https://www.etti.unibw.de/labalive/experiment/am/)

In Abbildung 4 lässt sich leicht erkennen, dass im Falle von $m = 0$ von einem modulationsfreien und von $m > 1$ von einem übermodulierten Signal die Rede ist.

3.1.2 Amplitudendemodulation

Bei bereits gehaltener Einfachheit der Modulation, fällt auch die Demodulation recht simple aus. Die wohl einfachste Verfahrenstechnik hierfür stellt die sogenannte Hüllkurvendemodulation dar.

Bei dieser wird das AM-Signal mittels eines Gleichrichters, beispielsweise durch eine Diode, auf eine einzelne Polarität begrenzt, wodurch nur noch die obere Hälfte der Hochfrequenzschwingungen verbleibt. Durch nachfolgenden Tiefpass lässt sich dann das hochfrequente Trägersignal entfernen und als Ergebnis verbleibt das ursprüngliche Modulationssignal. [8]

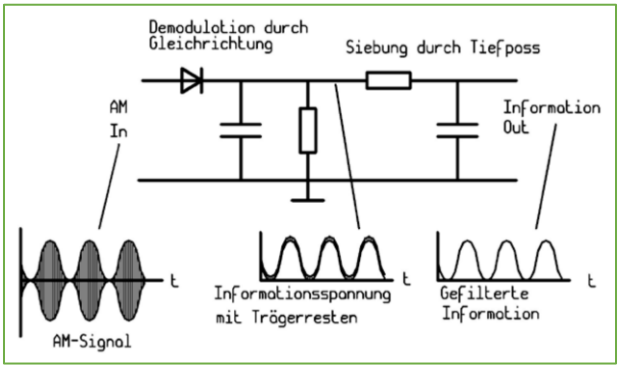


Abbildung 5 Skript Telekommunikationstechnik von Prof. Dr. -Ing Erwin Riederer, Seite 28, Abbildung 4-7

3.1.2 Frequenzmodulation

Die Frequenzmodulation, auch bekannt als „FM", ist eng verwandt mit der Idee der Amplitudenmodulation. Der entscheidende Unterschied besteht darin, dass bei der Frequenzmodulation nicht die Amplitude, sondern die Frequenz die veränderliche Größe des Modulationsverfahrens ist. Die Amplitude ist bei diesem analogen Modulationsverfahren konstant.

So variiert die Frequenz des Trägersignals proportional in Abhängigkeit zur Quellensignalamplitude. Eine einfache Realisierung kann über eine VCO – „voltage controlled oscillator“ erfolgen, dies ist in Abbildung 6 illustriert.

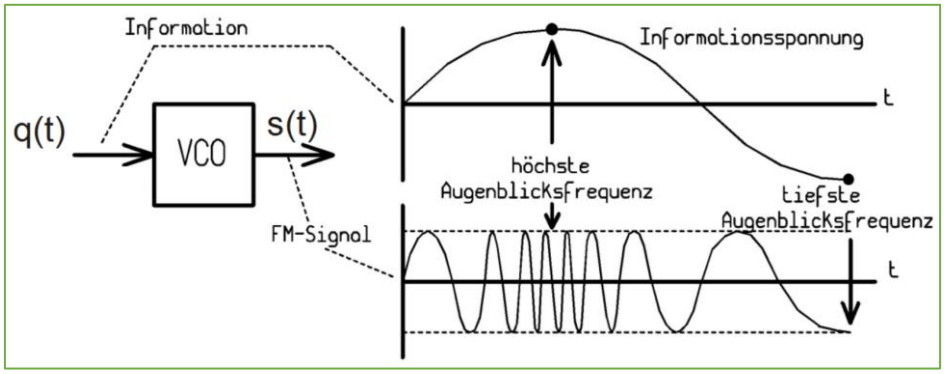


Abbildung 6 Skript Telekommunikationstechnik von Prof. Dr. -Ing Erwin Riederer, Seite 38, Abbildung 7-1

Die maximale Frequenzabweichung wird hierbei als sogenannter „Frequenzhub“ bezeichnet. Und die jeweils erzielte Frequenzabweichung von der Trägerfrequenz mathematisch als Produkt aus dem zu modulierenden Signal ($m(t)$) und der Modulationskonstante ($k_m = \left[\frac{V}{Hz}\right]$) definiert:

$$\Delta f(t) = k_m \times m(t)$$

Zusammenfassend zeichnet sich die Frequenzmodulation durch eine erhöhte Robustheit als AM aus, da die Information nicht mehr in der Amplitude des Signals enthalten ist. Dadurch haben eventuelle situative Amplitudendämpfungen, durch beispielsweises Fading, im Übertragungskanal keine Auswirkungen mehr auf die Qualität der Nachricht. Des Weiteren lässt sich über die Modulationskonstante k_m die benötigte Bandbreite variabel mit den Qualitätsanforderungen anpassen, in jedem Fall fällt der Bandbreitenbedarf jedoch höher als bei einem AM modulierte Signal aus und der Banderweiterungsfaktor J=2 stellt für FM die Untergrenze dar. Der Modulationsgewinn übertrifft mit beispielsweisen M = 0,75 jedoch deutlich denjenigen von AM und final lässt sich der FM eine bessere Signalqualität und Störfestigkeit als AM zusprechen. [9]

3.1.3 Frequenzdemodulation

In der Regel werden FM-Signale vor der Demodulation kurz um in AM-Signale oder, auf die in der Masterarbeit unbehandelte, Pulsmodulationssignale umgewandelt. Die reelle Schaltung, die die Umwandlung von FM in AM ermöglicht, wird als sogenannter Diskriminator bezeichnet und zählt zu den frequenzabhängigen Schaltungen.

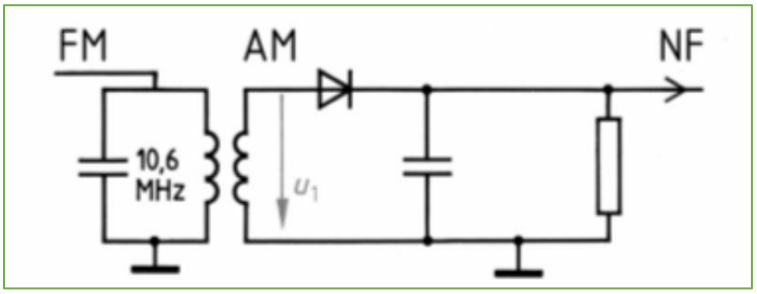


Abbildung 7 "Die Entstehung der Amplitudenmodulation aus Frequenzmodulation" Bild 12-26 (https://www.darc.de/der-club/referate/ajw/lehrgang-ta/a12/#FM_Demodulation)

Die Schaltung besteht hierbei erneut aus einem, wie in Kapitel „3.1.2 Amplitudendemodulation“ beschriebenen, Hüllkurvendemodulator, der nun jedoch mit einem vorangestellten Parallelschwingkreis ergänzt wurde. Der Parallelschwingkreis ergänzt die Fähigkeit, Frequenzänderungen in Amplitudenänderungen umzuwandeln, welches die Transformation eines FM-Signale in ein AM-Signal ermöglicht. Nach dieser Transformation wird die Demodulation gemäß der Hüllkurvendemodulator-Beschreibung aus Kapitel „3.1.2 Amplitudendemodulation" durchgeführt, wodurch letztendlich das ursprüngliche Modulationssignal zurückgewonnen wird. [10]

3.1.4 Einseitenbandmodulation

Die Einseitenbandmodulation, auch bekannt als „SSB" (engl. Single-Sideband Modulation), entstand aus der Bestrebung, die analoge Amplitudenmodulation spektrums- und energieeffizienter zu gestalten. Denn wie in Kapitel „3.1.1 Amplitudenmodulation" beschrieben, weist das herkömmliche Zweiseitenband-AM-Signal den Nachteil auf, dass die vollständige Information über das Quellensignal sowohl im USB als auch im LSB steckt. Das zweite Seitenband wird daher in der Nachrichtentechnik als redundant und als unnötiger Bedarf an Bandbreite angesehen.

Zur Überwindung dieses Problems wurde die Einseitenbandmodulation entwickelt. Mit dieser wird nur ein einziges Seitenband des AM-Signals übertragen, während das andere unterdrückt wird. Hierdurch wird die benötigte Bandbreite um die Hälfte reduziert und die Spektraleffizienz des Übertragungssystems verdoppelt. Mit dieser Maßnahme wird die Einseitenbandmodulation zu einer effizienten und praktischen Methode der Signalkodierung und Übertragung.

Damit jedoch aus einem Signal der Zweiseitenband-Amplitudenmodulation ein Signal der Einseitenbandmodulation wird, bedarf es einen höheren schaltungstechnischen Aufwand. In der Technik existieren hierfür unterschiedliche Verfahren, das in der Masterarbeit behandelte labAlive-Experiment der „Analog Demodulation“ setzt auf die Umwandlung mittels Hilbert-Transformation und geschickter Verschiebung der Phasen. Diese bildet aus einem reellen Signal ein analytisches Signal und verschiebt den Imaginärteil gegenüber dem Realteil um $\frac{\pi}{2}$ bzw. 90°. Werden diese beiden Anteile dann mit zwei phasenverschobenen Trägern gemischt und anschließend wieder addiert, so wird eines der beiden Seitenbänder ausgelöscht. [11] [12]

Mathematisch stellt sich diese wie folgt dar, wenn exemplarisch zur Berechnung in Abbildung 8 für $x(n) = cos(\mu n)$ gesetzt wird:

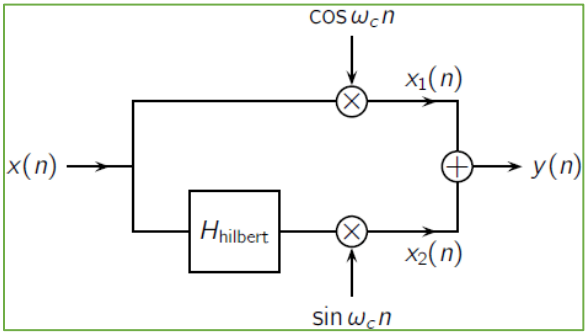


Abbildung 8 Single-Sideband Amplitude Modulation -MathWorks (https://de.mathworks.com/help/signal/ug/single-sideband-amplitude-modulation.html)

Im oberen Pfad, dem sogenannten Quadraturpfad, gilt nach dem Additionstheorem für $x_1(n)$:

$$\begin{aligned} \boldsymbol{x_1(n)} &= x(n) \times \cos(\omega_c n) \\ &= cos(\mu n) \times \cos(\omega_c n) \\ &= \frac{1}{2} \cdot [cos(\mu n - \omega_c n) + cos(\mu n + \omega_c n)] \end{aligned}$$

Im unteren Pfad hingegen, dem sogenannten In-phase Pfad, bewirkt die Hilbert-Transformation einen Phasenverschub für $x(n)$ um $\frac{\pi}{2}$. Hiermit gilt nach dem Additionstheorem für $x_2(n)$:

$$\begin{aligned} \boldsymbol{x_2(n)} &= x\left(n - \frac{\pi}{2}\right) \times \sin(\omega_c n) \\ &= cos\left(\mu n - \frac{\pi}{2}\right) \times \sin(\omega_c n) \\ &= \sin(\mu n) \times \sin(\omega_c n) \\ &= \frac{1}{2} \cdot [cos(\mu n - \omega_c n) - cos(\mu n + \omega_c n)] \end{aligned}$$

Bei der dann final erfolgenden Addition von $x_1(n)$ und $x_2(n)$ entsteht für $y(n)$:

$$\begin{aligned} \boldsymbol{y(n)} &= x_1(n) + x_2(n) \\ = \frac{1}{2} \cdot [cos(\mu n - \omega_c n) + cos(\mu n + \omega_c n)] &+ \frac{1}{2} \cdot [cos(\mu n - \omega_c n) - cos(\mu n + \omega_c n)] \\ &= \cos(\mu n - \omega_c n) \end{aligned}$$

Hierdurch lässt sich final zeigen, dass durch die Hilbert Transformation und geschickter Mischung mit zwei phasenverschobenen Trägern letztlich der Anteil „$cos(\mu n + \omega_c n)$“ durch die Addition von $x_1(n)$ und $x_2(n)$ entfernt wird. Folglich wird bei der Einseitenbandmodulation mit diesem Verfahren auch aus dem Zweiseitenband-AM-Signal das zweite redundante Seitenband erfolgreich entfernt beziehungsweise unterdrückt und damit, getreu dem Bestreben dieses Modulationsverfahrens, für ein spektrums- und energieeffizienteres Signal gesorgt.

3.1.5 Einseitenbanddemodulation

Aufgrund dieser Signalbearbeitung reicht es für die Demodulation nicht aus, wie für ein konventionelles AM-Signal einen Hüllkurvendemodulator zu verwenden.

Stattdessen wurde hierfür das sogenannte Verfahren der Synchrondemodulation entwickelt, welches obendrein auch als Demodulator eines Zweiseitenbandsignales verwendet werden kann. Dieses mischt, wie auch schon senderseitig im Modulator, das Signal mit einer vom lokalen Oszillator erzeugten Trägerfrequenz, beispielsweise durch einen VCO. Der Begriff „Synchrondemodulation“ ergründet sich dabei darin, dass der empfangsseitige lokale Oszillator die gleiche Trägerfrequenz wie der senderseitige Oszillator der Modulation aufweist. Dies wird in Abbildung 9 ersichtlich, in der dieselbe ω_T-Trägerfrequenz sowohl im Modulator als auch im Demodulator beigemischt wird. Es sei jedoch betont, dass die Abbildung Bezug auf ein Zweiseitenbandsignal nimmt, der Einsatz des Synchrondemodulator und dessen Prinzip ist jedoch identisch.

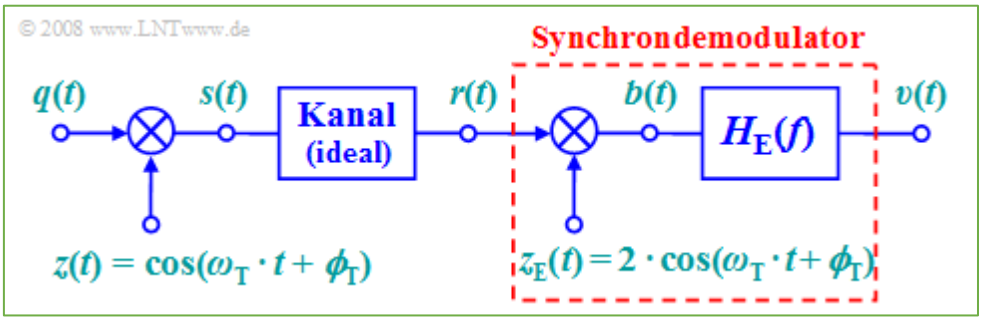


Abbildung 9 "ZSB–Amplitudenmodulation und Synchrondemodulation" (https://www.lntwww.de/Modulationsverfahren/Synchrondemodulation)

Aus dieser Operation gehen aus dem ursprünglichen eingangsseitigen Einseitenbandsignal zwei Nutzsignalspektren hervor, das eine damit in Basisbandlage und das andere bei doppelter Trägerfrequenz.

Zum Abschluss kann das ursprüngliche Informationssignal mithilfe einer Tiefpassfilterung aus dem Signal der Basisbandlage zurückgewonnen werden.

Schaltungstechnisch wird dies erreicht, indem Analogschalter das Einseitenband aus dem zu demodulierenden Signal periodisch im Takt des synthetisierten Trägers schalten oder abtasten. [13]

3.1.6 Quadraturamplitudenmodulation

Die Quadraturamplitudenmodulation, kurz „QAM“, folgt erneut der Bestrebung die Amplitudenmodulation spektrums- und energieeffizienter zu gestalten. Jedoch zählt sich die QAM in der Fachliteratur überwiegend zu den digitalen Modulationsverfahren, wohlgleich sie neben der gleichzeitigen Übertragung von zwei digitalen Bitströmen auch zur gleichzeitigen Übertragung von zwei analogen Nachrichtensignalen eingesetzt werden kann.

Die Grundidee der QAM beruht auf der geschickten Nutzung der Orthogonalität der Cosinus- und Sinusfunktionen. Dabei werden zwei unabhängige Quellensignale mit Trägersignalen gemischt, die um 90° phasenverschoben zueinanderstehen. Durch diese Mischung können die Signale gleichzeitig über einen Kanal gesendet werden, ohne sich gegenseitig zu beeinträchtigen.

Im Vergleich zur herkömmlichen Amplitudenmodulation kann das Frequenzband damit doppelt genutzt werden (doppelte Spektrumseffizienz). [14]

Abbildung 10 zeigt die beschriebene Schaltung der Modulation:

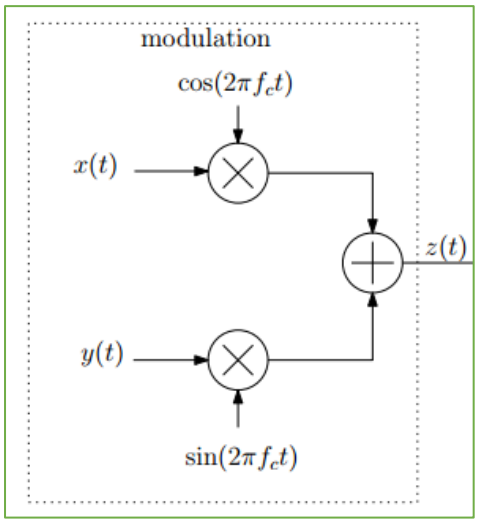


Abbildung 10 block diagram of quadrature amplitude modulation (https://www.chegg.com/homework-help/questions-and-answers/quadrature-amplitude-modulation-qam-commonly-used-modulation-scheme-communication-systems--q3960111)

Getreu dieser Abbildung stellt sich das zu übertragene Signal $z(t)$ folgendermaßen dar:

$$\boldsymbol{z(t)} = x(t) \times \cos(2\pi f_c t) \; + \; y(t) \times \sin(2\pi f_c t)$$

Da gilt „$\cos(xt) = \sin\left(xt + \frac{\pi}{2}\right)$“, weisen die beiden zugemischten Trägersignale den zuvor angeführten Phasenverschub um 90° auf und versetzen die beiden Quellensignale in eine zueinander gerichtete Orthogonalität, damit sich die Summensignalanteile bei gleichzeitiger Übertragung in $z(t)$ nicht beeinträchtigen.

Aufgrund des Phasenverschubes wird der obere Pfad in Abbildung 10 daher auch als „In-phase Pfad“ und „In-phase-Anteil“ und der untere Pfad als „Quadraturpfad“ und „Quadratur-Anteil“ bezeichnet beziehungsweise in ihrer Gesamtheit als Operation als IQ-Verfahren.

Des Weiteren lassen sich aus dem In-phase- und Quadratur-Anteil Informationen über die Amplitude und Phase ableiten:

$$A(t) = \sqrt{I(t)^2 + Q(t)^2}$$

$$\varphi(t) = arctan\left(\frac{Q(t)}{I(t)}\right)$$

Das zu übertragene Signal $z(t)$ lässt sich daher auch anhand dieser Informationen beschreiben:

$$z(t) = A(t) \cdot cos\big(2\pi f_c t + \varphi(t)\big)$$

In der IQ-Ebene lassen sich dadurch verschiedene Symbolpunkte bilden, von denen jeder durch eine eindeutige Kombination aus Amplituden- und Phaseninformation festgelegt ist. Dies ist in Abbildung 11 dargestellt.

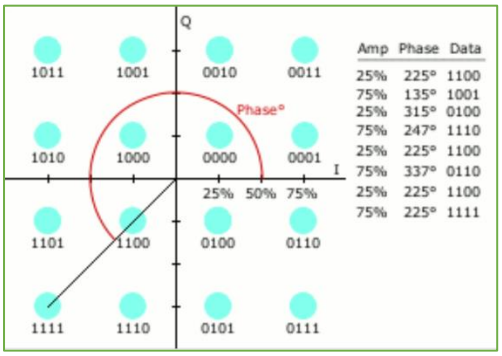


Abbildung 11 16-QAM Demonstration
(https://de.wikipedia.org/wiki/Datei:16-QAM_Demonstration.gif)

3.1.7 Quadraturamplitudendemodulation

Zur Demodulation wird das in Abbildung 10 gezeigte System erweitert und $z(t)$ jeweils erneut mit dessen Trägersignalen aus dem Modulationsprozess gemischt.

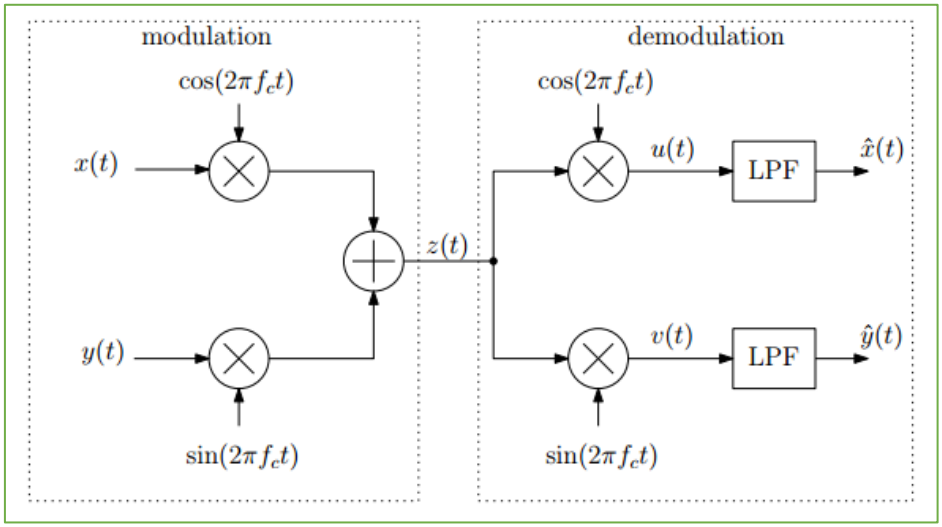


Abbildung 12 block diagram of quadrature amplitude modulation and demodulation (https://www.chegg.com/homework-help/questions-and-answers/quadrature-amplitude-modulation-qam-commonly-used-modulation-scheme-communication-systems--q3960111)

Für den oberen Pfad des Demodulators ergibt sich damit für $u(t)$ folgende mathematische Gleichung:

$$\begin{aligned}
\boldsymbol{u(t)} &= z(t) \times \cos(2\pi f_c t) \\
&= [x(t) \times \cos(2\pi f_c t) \; + \; y(t) \times \sin(2\pi f_c t)] \times \cos(2\pi f_c t) \\
&= x(t) \times \cos(2\pi f_c t) \times \cos(2\pi f_c t) \; + \; y(t) \times \sin(2\pi f_c t) \times \cos(2\pi f_c t) \\
&= \frac{1}{2} x(t) \times [1 + \cos(4\pi f_c t)] \; + \; \frac{1}{2} y(t) \times \sin(4\pi f_c t) \\
&= \boldsymbol{\frac{1}{2} x(t)} + \frac{1}{2}[x(t) \times \cos(4\pi f_c t) + y(t) \times \sin(4\pi f_c t)]
\end{aligned}$$

Durch diese Operation entstehen ein separiertes und im Basisband liegendes Quellensignal $x(t)$ und zusätzliche Mischprodukte mit doppelter Frequenz.

Die Quellensignalinformation $x(t)$ kann dabei nun vollständig rekonstruiert werden, wenn mithilfe eines Verstärkers der Faktor $\frac{1}{2}$ kompensiert und durch einen „LPF“ (engl. lowpass filter = Tiefpassfilter) die entstandenen Mischprodukte bei doppelter Frequenz weggefiltert werden.

Das Signal $\boldsymbol{v(t)}$ des unteren Pfades lehnt sich an diese Herleitung an:

$$\begin{aligned}\boldsymbol{v(t)} &= z(t) \times \sin(2\pi f_c t) \\ &= [x(t) \times \cos(2\pi f_c t) \; + \; y(t) \times \sin(2\pi f_c t)] \times \sin(2\pi f_c t) \\ &= x(t) \times \cos(2\pi f_c t) \times \sin(2\pi f_c t) + \; y(t) \times \sin(2\pi f_c t) \times \sin(2\pi f_c t) \\ &= \frac{1}{2} x(t) \times \sin(4\pi f_c t) + \frac{1}{2} y(t) \times [1 - \cos(4\pi f_c t)] \\ &= \boldsymbol{\frac{1}{2} y(t)} + \frac{1}{2}[\, y(t) \times (-\cos(4\pi f_c t)) + x(t) \times \sin(4\pi f_c t)]\end{aligned}$$

Nach obiger Beschreibung kann hier ebenso das zweite Quellensignal $y(t)$ vollständig rekonstruiert werden.

Durch diese Signaloperationen lässt sich das Frequenzband, wie gezeigt, ohne Beeinträchtigung doppelt nutzen.

3.2 Weißes gaußsches Rauschen

Das „weiße gaußsche Rauschen“, auch AWGN (engl. *additive white Gaussian noise*) genannt, beschreibt eine Rauschgröße, in der zwei charakteristische Eigenschaften gemeinsam auftreten. Denn als reines „weißes Rauschen“ wird in der Theorie der Kommunikationstechnik ein Rauschsignal mit konstanter spektraler Rauschleistungsdichte verstanden ($S(f) = const.$). Also einem Rauschsignal, dass, wie in Abbildung 13 gezeigt, über alle Frequenzzahlen hinweg über die gleiche Rauschleistung verfügt.

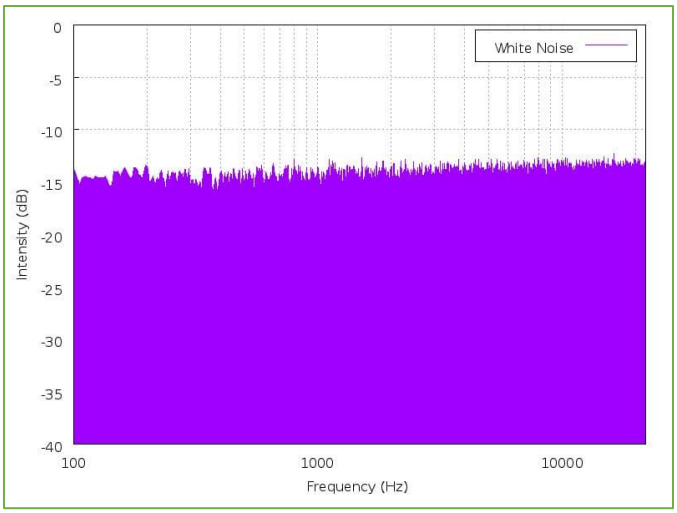


Abbildung 13 Spektrumsansicht des weißen Rauschens (https://www.audiotechnology.com/tutorials/on-the-bench-whats-that-noise)

In der Realität kann dies jedoch nur abschnittsweise auf einem definierten und begrenzten Frequenzbereich auftreten, da dieses Signal sonst über eine unendlich hohe Leistung/Energie verfügen müsste.

Als reines „gaußsches Rauschen“ wird hingegen ein Rauschsignal mit einer Signalamplitudenverteilung verstanden, dessen Dichtefunktion der gaußschen Glockenkurve beziehungsweise der Standardnormalverteilung entspricht. Diese ist exemplarisch in Abbildung 14 dargestellt.

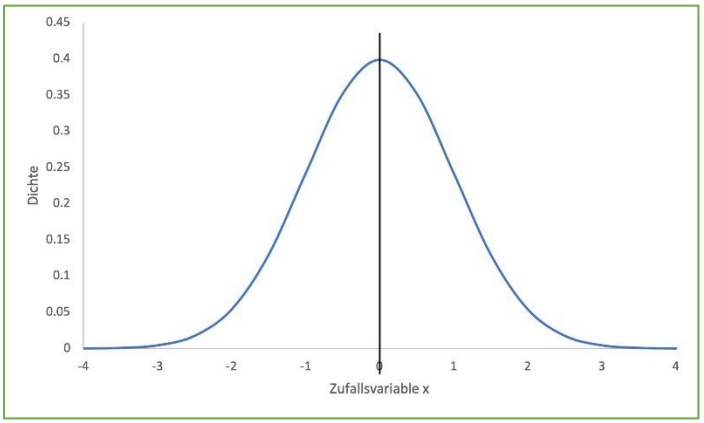


Abbildung 14 Dichtefunktion der Gaußschen Glockenkurve (https://www.lektorat-plus.de/posts/normalverteilung.php)

Das „weiße gaußsche Rauschen“ in seiner Gesamtheit definiert demnach ein Rauschsignal, welches beide charakteristischen Eigenschaften in einem Signal vereint. [15]

4. Durchführung

4.1 MyLabAlive – Ausgangslage der Dokumentationen

Wie bereits in der Einleitung vorweggenommen, beinhaltet das Online-Labor „labAlive“ die Nutzerumgebung „myLabAlive“. In diesem kann der Anwender nach seinen eigenen Vorstellungen eigene Java-Anwendungen erzeugen, ganz nach dem Vorbild des bereits bestehenden Experimenten-Pools der Plattform. Damit der Nutzer sich jedoch mit dieser Funktionalität einfinden kann, benötigt es eine Einführung und Dokumentation, die hierzu assistiert.

Unter der Adresse „https://www.etti.unibw.de/labalive/my/“ fand sich bereits eine solche Urversion einer Dokumentation, jedoch ist deren Nutzen als einleitende Hilfestellung nur bedingt gegeben.

So kam diese seither eher einer Sammlung an Beispielen und Vorschlägen zum Nachahmen gleich, statt einer richtigen Anleitung und Vorstellung der Funktionen. Ein Eindruck dieser Sammlung mit insgesamt zuvor 106 Showcases findet sich aufgrund der Dimension in Anhang 1. Des Weiteren wurden App-Einstellungen wie die Diagramm-Settings behandelt, die aufgrund der „User changes“-Erzeugung so in ihrer Form nur überschaubar beim Nutzer für eine initiale Text2App Verwirklichung Anwendung finden, dazu später mehr im Kapitel 4.4 bei den Inhalten für die Videocliperzeugung.

Als Verbesserung hierzu bestand versteckt bereits eine zweite Version unter dem Link „https://www.etti.unibw.de/labalive/my2/“ allerdings war bei dieser zu bemängeln, dass trotz der anschaulicheren Grafiken die erklärenden Texte missverständlich und der redaktionelle Seitenaufbau oftmals irreführend blieb. So gingen beispielsweise Inhalte durch die Beschreibungstexte unter und Funktionserklärungen tauchten über die Unterseiten hinweg mehrfach (redundant) auf.

Unter Berücksichtigung dieser Gesichtspunkte bestand die Aufgabe darin, eine veröffentlichungsfähige Dokumentation für „myLabAlive“ zu generieren, die die Inhalte der Urversion abdeckt, allerdings jedoch prägnanter und zielführender gestaltet, und sofern passend, die Grafiken der Beta-Version der my2-Variante mit ihren Unterseiten „myLab“, „myExample“, „Your App“, „Safe my Work“, „Layout Creation“ und „JavaDoc Index“ als Inspiration aufgreift.

Im Folgenden wird nun dieser überarbeitende Arbeitsprozess, die Hintergründe und die erzielten Ergebnisse beschrieben.

4.2 MyLabAlive – Überarbeitete/Neue Tutorialeinführung „myLab“

Den Beginn dieser Überarbeitung machten die Arbeiten an den einleitenden Tutorialunterseiten, welche jeweils als erster Kontaktpunkt zwischen dem Nutzer und der assistierenden Anleitung zu verstehen ist.

Eine Gemeinsamkeit, die bei beiden der vorherigen Varianten bestand, ist das Fehlen eines Einstieges, der in wenigen Sätzen eine kurze Beschreibung über den folgenden Inhalt liefert.

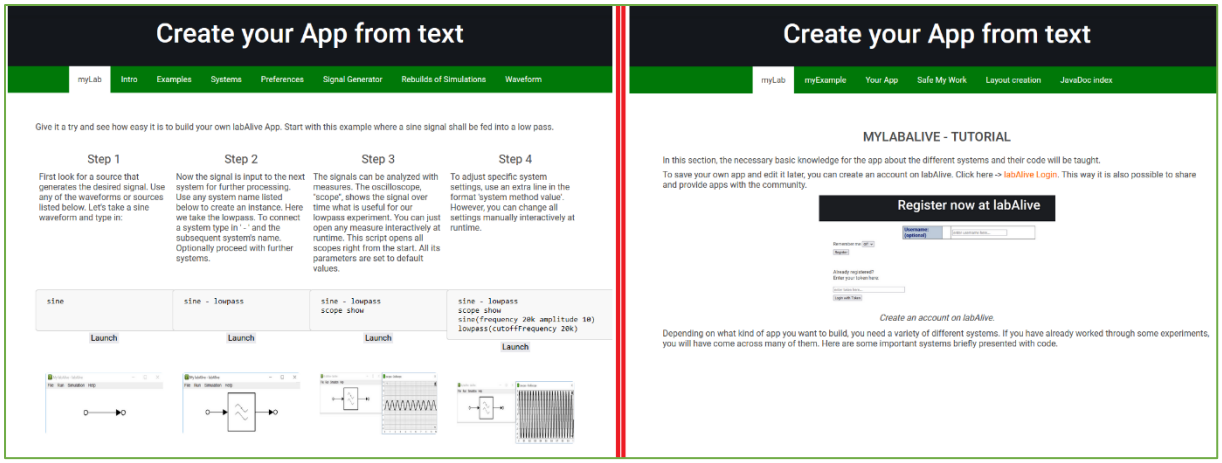


Abbildung 15 MyLabAlive Tutorialeinführung der alten Versionen (links: Urversion, rechts: my2-Version)

Wie in Abbildung 15 ersichtlich wird, fehlt in der Urversion in Gänze ein genereller thematischer Einstieg, hingegen in der my2-Variante der eigentliche Kern des Tools verschwiegen blieb und stattdessen ein veralteter Login und Speicherungsprozess, der eigenen Appgenerierung beschrieben wurde. Problem hierbei war es, dass sich nicht nur der Login/Anmeldeprozess mit der Arbeit der diesjährigen Bacheloranden verändert hatte und damit veraltet war, sondern wie am obigen Reiter zu erkenne ist, ohnehin sich ein eigenes Kapitel diesem Thema widmete und eine vorweggenommene Mehrfachnennung damit redundant ist.
Gelöst wurde dies mit folgendem Schritt, dessen Ergebnis in Abbildung 16 dargestellt ist: Um dem Nutzer sowohl einen animierenden als auch klaren Einstieg zu ermöglichen, wurde der Kerninhalt des Tools in Form eines kurzen Beschreibungstextes festgehalten. Zur illustrativen Veranschaulichung wurde obendrein ein nicht ausführbares Showcase-Example eingefügt, um dem Anwender eine erste Idee der Aufmachung zu vermitteln.

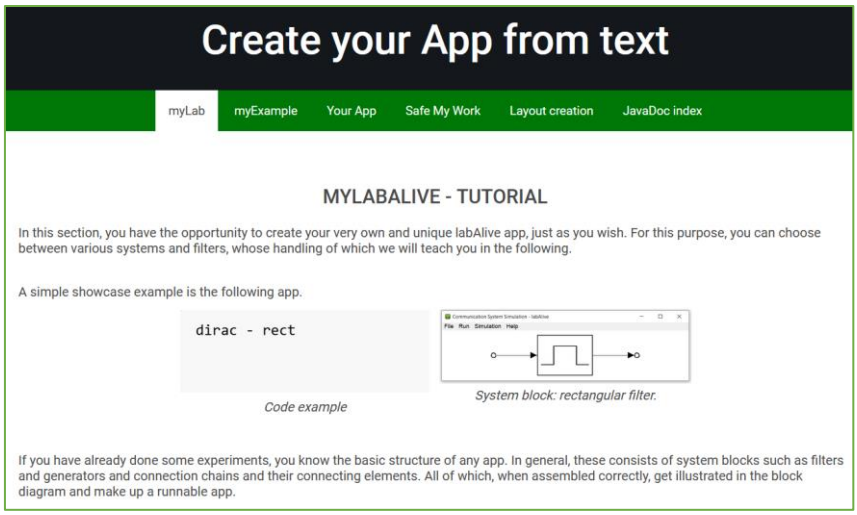


Abbildung 16 MyLabAlive neue Tutorialeinführung

Folgend zur Einführung wird im letzten Absatz in Abbildung 16 textbasiert das Glossar benannt, an dem sich die weiterführende Anleitung orientiert. Hierin finden sich auch die Schlüsselwörter der allgemeinen CT-Elemente. Eine ausführbare ***App*** besteht demnach aus ***system blocks***, wie Filter und Generatoren, die mit ***connection chains*** und deren ***connecting elements*** verknüpft werden. Bei richtiger Programmierung finden sich die Elemente dann in einem sich generierten ***block diagramm*** wieder, getreu der Layout-Anordnung durch die Programmierbefehle.

Als weiteres wichtiges Element der Einführung für einen umfassenden Überblick ist die weiterführende Betrachtung der Begriffe aus dem Glossar. Getreu des kommunikationstechnischen Aufbaus eines Systems wurde die Einführungsreihenfolge aus der damaligen my2-Dokumentationsversion verändert und nun als erstes die Signalquelle statt wie zuvor die Verbindungselemente thematisiert. Dem Nutzer werden hierbei die gängigen Source-Optionen und die zur Verfügung stehenden Signalwellenformen aufgezeigt. Aus Gründen der Übersichtlichkeit und Veranschaulichung wurde sich für die Beibehaltung der „Slideshow“-Programmierung aus der ehemaligen my2-Version entschieden. In dieser werden die Signalwellenformen Sinus, Kosinus, Dreieck, Sägezahn, Rechteck, Gleichstrom (DC), Laplace-Verteilung und Dirac Kamm mit ihren Signalverläufen gezeigt und eingeführt. In jener Reihenfolge, in der sie auch bei laufender App dem Nutzer im Drop Down Menü des Signalgenerators vorgeschlagen werden. Dies ist in Abbildung 17 ersichtlich.

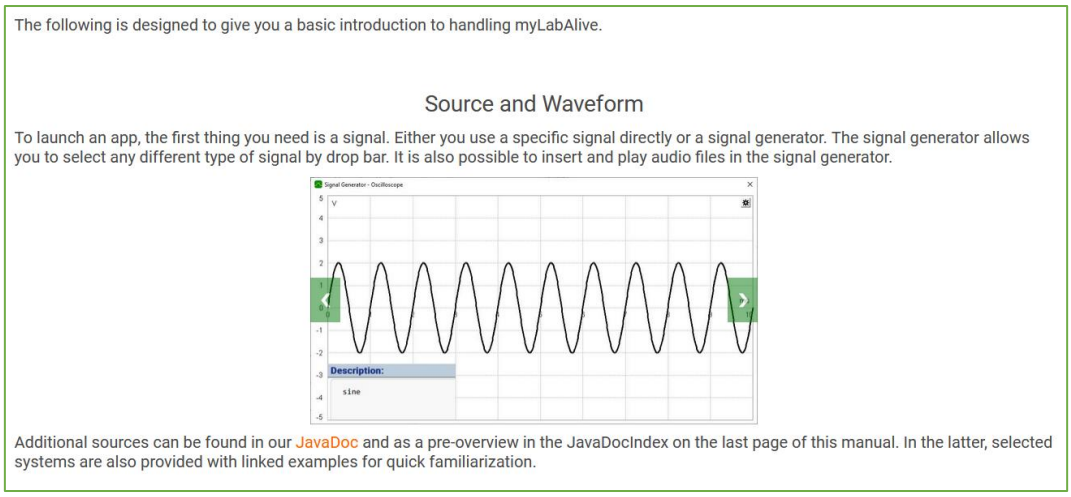


Abbildung 17 "Source and Waveform" MyLabAlive neue Tutorialeinführung

Da in LabAlive mehr Quellen als nur die hier vorgestellten gängigsten existieren, findet sich am Ende noch eine Verlinkung zu JavaDoc und eine Erwähnung einer alternativen beispielgestützten Überblickstabelle (hierzu mehr im Kapitel „4.6 JavaDoc Index“).

Nach der Einführungsbeschreibung der Quellen werden die zur Verfügung gestellten Systeme und deren Verbindungselemente behandelt (siehe Abbildung 18). Der Begriff „Systeme“ wird hierbei noch einmal separat definiert und exemplarisch mit einer Tiefpass-Filter App untermalt. Aufgrund der Masse an Systemen wäre es jedoch unübersichtlich alle Systeme in der Einführung mit aufzulisten. Stattdessen findet sich auch hier wie bereits bei den Sources eine Verlinkung zur Dokumentation JavaDoc und eine Ankündigung auf die als Unterseite am Ende der Anleitung folgenden Übersichtstabelle mit den gängigsten Systemblöcken.

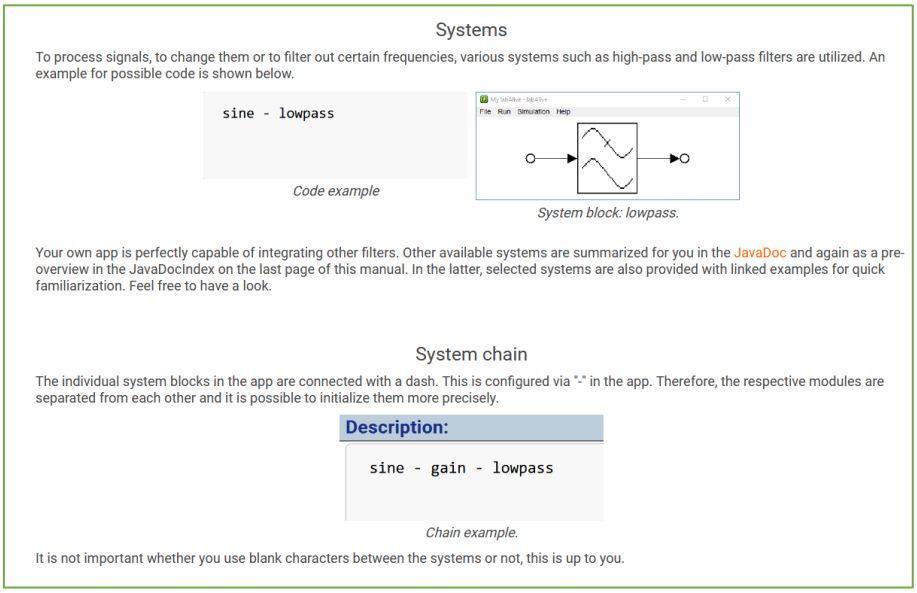


Abbildung 18 "Systems und System Chain" MyLabAlive neue Tutorialeinführung

Abschließend als letztverbleibendes Grundelement wird das reine Verbindungselement beschrieben. Hiermit wurden dem Anwender sämtliche notwendigen Elemente einer einfachen lauffähigen App vorgestellt.

Bei umfassenderem Umgang wird jedoch die Notwendigkeit von vernetzten Apps ersichtlich. Multiple Systeme, Adder, Splits und Multiplier werden gebraucht und damit ebenso in der Einführung aufgegriffen.

Aus der ehemaligen my2-Dokumentationsversion konnten sich hierfür anschauliche Grafiken übernehmen lassen, jedoch wurden deren Beschreibungstexte erneut abgewandelt und verdeutlicht. Die Grafiken sind erneut in einer Diashow eingebettet, in der diesmal Adder und Multiplier im block diagramm ersichtlich werden und zwei synchronisierte GIFs, die eine sich aufbauende Schaltung multipler Systeme zeigen. Diese sind in Abbildung 19 in ihrer finalen Form abgebildet.

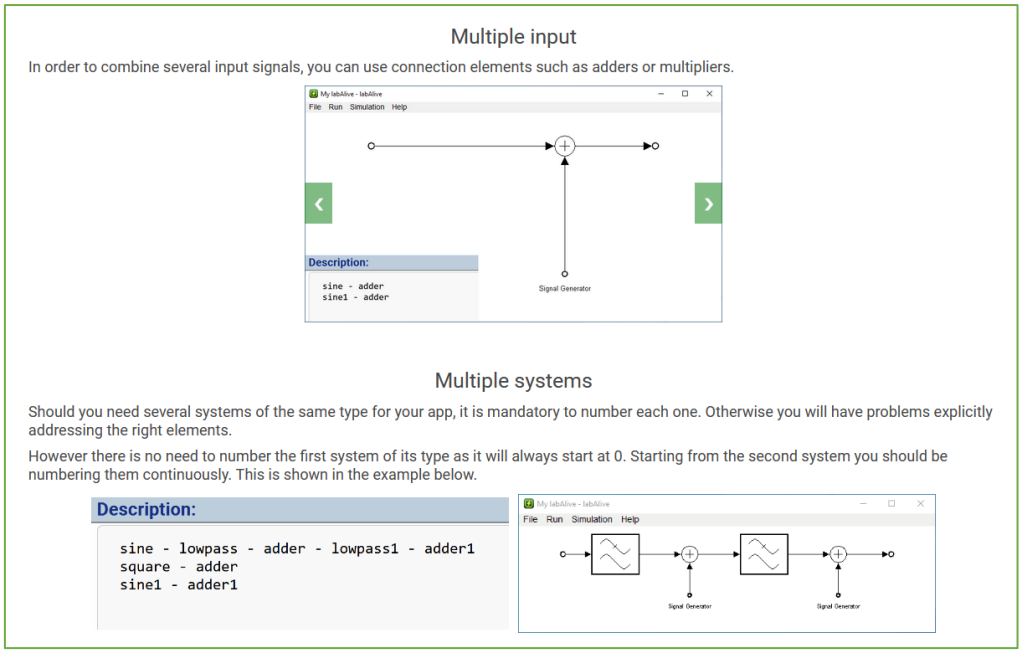


Abbildung 19 "Multiple input/-systems" MyLabAlive neue Tutorialeinführung

Abschließend in der Einführung der Anleitung wird als letztes Element auf die beiden verschiedenen verfügbaren Programmierstile hingewiesen, zwischen denen der Nutzer frei wählen kann. Denn Systemparameter können auf zwei verschiedene Arten angepasst werden: zum einen durch die Verwendung eines Methodenaufrufs/Setter, zum anderen durch einen "Shortcut Setter" und den Konstruktor. Zur Veranschaulichung dieser werden auch hier dem Nutzer Grafiken aus dem Programmcode präsentiert und mit einem Beschreibungstext die Unterschiede beider Programmierstile hervorgehoben. Zusätzlich hierzu finden sich am Ende der Einführungsseite erste Beispiele, deren stilistische Verwendung auf die Folgeseite der „myExample“ überleitet. Diese entstammen jedoch aus der ehemaligen my2-Version und entziehen sich dem eigenen Tätigkeitsbereich.

4.3 MyLabAlive – Überarbeitung des Beispielabschnittes „myExample“

Während die einleitende Unterseite „myLab“ dem Nutzer lediglich einen ersten allgemeinen Überblick über die App-Bestandteile bot, wird ihm/ihr auf der Folgeseite „myExample“ erstmalig in der Anleitung die Möglichkeit geboten, anhand ausgewählter Beispiele die komplette Funktionalität von myLabAlive interaktiv einzustudieren.

Wie im Kapitel „4.1 MyLabAlive – Ausgangslage der Dokumentationen“ bereits vorweggenommen, besaß die ursprüngliche Anleitung von „myLabAlive“ eine Beispielsammlung von 106 Aufgaben, in der sich der Nutzer selbst hat zurechtfinden und sich seine ersuchten Informationen heraussuchen müssen. In der ersten nachgearbeiteten Version „my2“ wurde diese bereits vorläufig zu einem gewissen Ansatz aussortiert, illustriert und damit verbessert.

Bei der Aufnahme der Arbeiten fiel jedoch auf, dass im Vergleich zur Urversion ein Teil der Funktionen in der my2-Anleitungsversion herausgelassen oder vergessen wurden und diese damit unvollständig war. Erste Handlungsschritte widmeten sich also dem Vergleich des Funktionspools beider Versionen. Hierbei wurden die 106 damaligen Showcase-Beispiele katalogisiert, in ihrer Zweckmäßigkeit als vertiefende Übungsaufgaben beurteilt und letztlich mit dem Bestand aus my2 abgeglichen und bei fehlender Abdeckung in dieses überführt.

Die erbrachte Überarbeitung nahm dabei diese Form an:

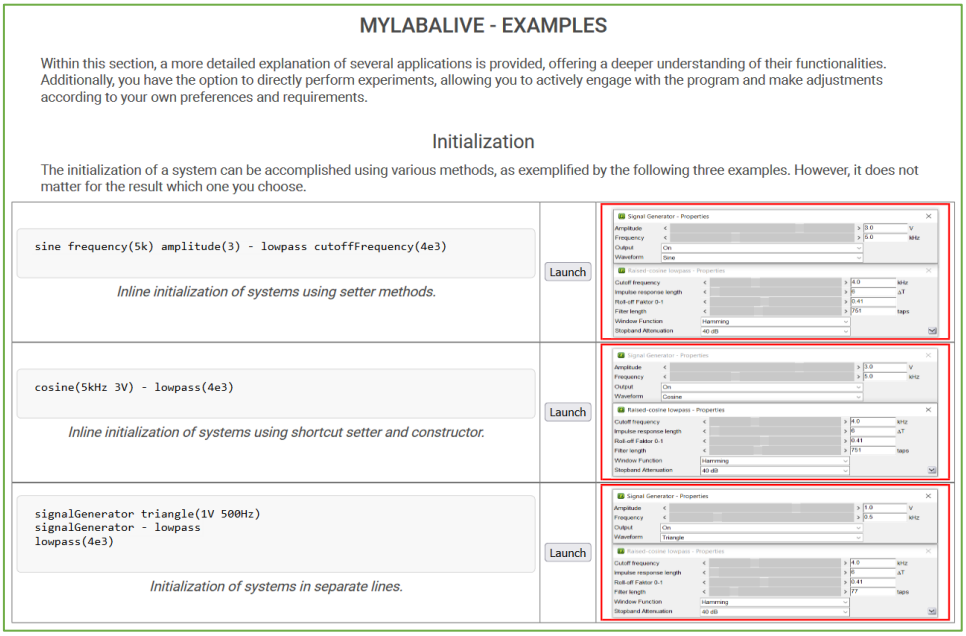


Abbildung 20 "Intro und Initialization - myExample" MyLabAlive neue Funktionsanleitung

Abbildung 20 zeigt den Beginn der Anleitung zu den Funktionen durch eigene Beispiele und zugleich eines von sieben katalogisierten Funktionsoberthemen. Um dem Nutzer einen vorherigen Überblick über die im folgenden behandelten Inhalte zu vermitteln, wurde erneut klassisch ein Intro verfasst.

Der Einstieg in das Thema der Funktionen erfolgt konsequent mit der zuvor im vorherigen Kapitel beschriebenen Initialisierung mittels unterschiedlicher Programmierstile, wobei diese jedoch auch interaktiv maschinell und automatisiert von der laufenden App erbracht werden können. Dies wird in Form eines eigens hierfür erstellten Videos präsentiert, dessen Entstehung und Inhalt aus Gründen der Übersichtlichkeit und Umfang jedoch erst in Kapitel „4.4 MyLabAlive – Videoerstellung für Beispielabschnitt“ beschrieben wird.
In der myLab-Tutorialeinführung, also der ersten Anleitungsunterseite, wurden bereits Showcase Beispiele des Programmierstils mittels Setter und Konstruktor aufgezeigt. Also die Möglichkeit, die Systemblockeinstellungen entweder durch dedizierte Methodenaufrufe und unter Angabe des gewünschten Parameterwerts zu verändern oder verkürzt durch die Nutzung des Konstruktors, bei der es ausreicht den Parameterwert direkt bei der Nennung des Systems zu hinterlegen.
Neu in diesem Abschnitt ist es nun, dass neben der Präsentation dieser auch vertiefend Illustrationen der Auswirkungen auf Appoberfläche gezeigt werden und die Programmcodes über den Launch-Button ausführbar sind. Dies soll zum Ausprobieren animieren, damit sich der Nutzer interaktiv mit der Funktionalität vertraut machen kann. Die in der Farbe Rot eingerahmten Illustrationen spannen sich hierbei automatisch zur besseren Lesbarkeit auf, sobald der Mauscursor diese überläuft.

Die nächstbehandelte Funktionalität umfasst das „Labeln“. Diese machte in der Ur-Anleitungsversion den Beginn und zeigte dem Nutzer die Möglichkeit auf, selbstgewählte Namen an Systeme zu vergeben, um diese bei späteren Arbeiten gezielter wiederzufinden. Die Aufmachung und Illustrierung dessen gleicht dem Abschnitt des Funktionsoberthemas „Initialization“ aus Abbildung 20.
Gefolgt wird diese vom Abschnitt der „Connection Line“. Hier wird das Verbinden einzelner Systemblöcke behandelt und zugleich das Vernetzen von Schaltungen mit multiplen Ein- und Ausgängen.
Für die Heranführung des Nutzers fiel hier die didaktische Entscheidung auf eine sich grafisch Schritt für Schritt aufbauende Beispiel-App, damit die Funktionalität klar ersichtlich bleibt und das Verbindungselement, hier ein Adder, für die Mehrwege-Schaltungen gesondert beleuchtet werden kann. Dies wird in Abbildung 21 ersichtlich.

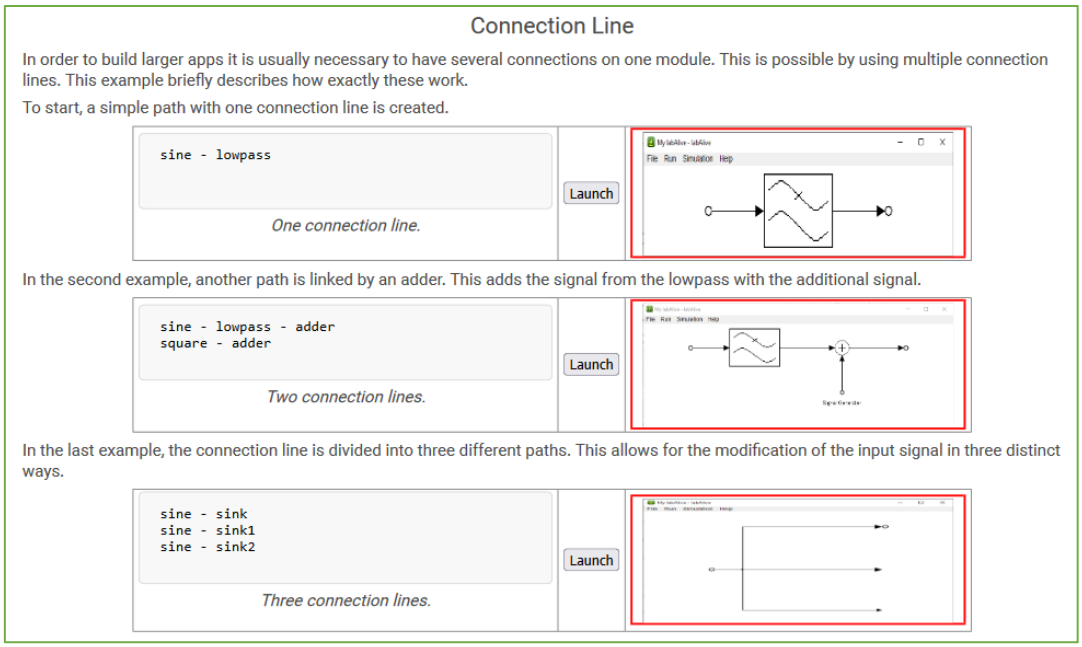


Abbildung 21 "Connection Line - myExample" MyLabAlive neue Funktionsanleitung

Als viertes von sieben Funktionsoberthemen findet sich die Vorstellung der „Scope Settings“. In diesem soll der Nutzer mit den Programmierbefehlen für das virtuelle Oszilloskop vertraut gemacht werden. Diese umfassen eine Spanne von der grundlegenden Rastereinstellung der Zeit- und Amplitudenachse, über die Einstellung der Oszilloskop Auflösung und Pixeldichte bis hin zum grafischen Stil des Signalgraphen.

Diese Einstellungen können bei der Appgenerierung zwar selbst als Simulationsbeschreibung eingegeben werden, jedoch lassen sich diese Einstellung auch eleganter und einfacher interaktiv in der App tätigen.

Denn die vom Nutzer ausgeführte App bietet nicht nur die Möglichkeit, ihre Parameterwerte während der Simulation fließend anzupassen, sondern besitzt auch eine zusätzliche Speicherfunktion. Die hierdurch während der Ausführung der Simulation gespeicherten Einstellungen werden dem Nutzer in Form eines fertigen Programmcodes übergeben, einschließlich der beispielsweisen Konfigurationen des virtuellen Oszilloskops. Dadurch erhält der Nutzer die Flexibilität, seine individuellen Einstellungen für zukünftige Simulationen beizubehalten und auf einfache Weise wiederzuverwenden. Die Speicherfunktion bietet eine praktische Lösung, um die Konfiguration der App effizient zu tätigen und Zeit bei der erneuten Eingabe von Parametern zu sparen. Im Tutorialtext der Scope Settings wird deshalb auf die elegantere interaktive Variante verwiesen, welche vom Video am Beginn der Unterseite abgedeckt wird.

Die Einstellungen, unabhängig vom gewählten Generierungsweg, werden für den Anleitungsabschnitt in Abbildung 22 als fertiger Programmcode wie folgt vorgestellt:

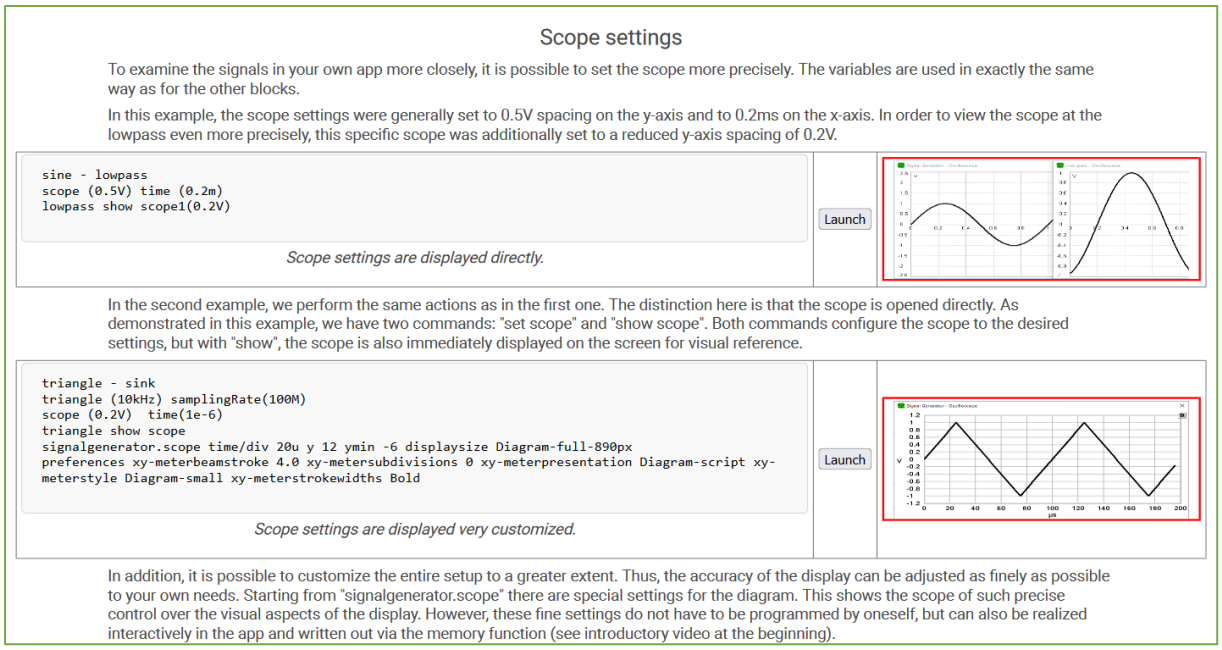


Abbildung 22 "Scope settings - myExample" MyLabAlive neue Funktionsanleitung

In diesem Abschnitt werden die zuvor benannten Funktionen aufgegriffen und, wie zuvor, erneut in Form einer ausführbaren App dem Anwender zum Experimentieren und Einstudieren zur Verfügung gestellt.

Womit es nahtlos übergeht, zu den letztverbleibenden drei Themenabschnitten der Beispielseite, diese wurden aufgrund der angezielten Vollständigkeit eingebracht, fallen jedoch aus dem im folgenden genannten Gründen allesamt kürzer aus. Diese umfassen zum einen das Thema der personalisierten Layoutanpassung für die eigene generierte Schaltung, sowie zwei Unterthemen, das der „enum Variable“ und das der „String function“.

Der Beispielabschnitt der Layoutanpassung ist als knapp gehaltener Teaser zu interpretieren, da aufgrund des inhaltlichen Umfangs dieses Themenbereichs eine eigenständige vertiefende Anleitungsunterseite generiert wurde (siehe Kapitel 4.5).
Deshalb ist hier im Beispielabschnitt lediglich eine einzelne ausführbare App vorhanden. Diese App dient dazu, die allgemeinen Zeichen als Layoutbefehle erstmalig aufzuzeigen und verweist über den Beschreibungstext auf die umfangreichere separate Anleitungsunterseite hin. Abbildung 23 zeigt diesen zu verstehenden Teaser samt dem kurzen Beispiel:

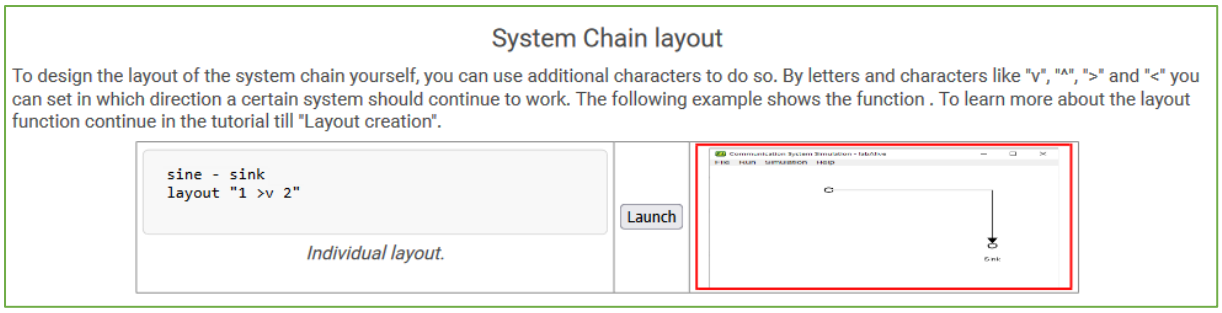


Abbildung 23 "System Chain layout -myExample" Kurzbeispiel mit Verweis auf eigene Unterseite

Die zwei verbleibenden Unterthemen, das der „enum variable“ und der „String function“ sind ebenso kurzgehalten, da sie in ihrer Anwendungshäufigkeit im Vergleich zu den vorangegangenen Funktionalitäten deutlich überschaubarer ausfallen. Denn deren Funktionalität ist eher im Bereich der Sonderbefehle einzuordnen.

So kann die Enum Variable in ihrer aktuellen Fassung als Konstante für vier Sonderfälle genutzt werden:

Als Konstante einer Leistungsdimensionierung des weißen gaußschen Rauschens („`EQUAL POWER WHITE NOISE` “), als Energiekonstante der Impulsantwort („`IMPULS RESPONSE ENERGY` “), als Konstante für ein Tiefpassspektrum („`LOWPASS SPECTRUM` “) und als Konstante für eine maximale Impulsantwort („`MAX IMPULS RESPONSE` “).

Zur Präsentation dieser Funktionalität werden dem Nutzer zwei Enum Variablen mit deren Verwendung vorgeführt. Vor allem ist hier auf die richtige Syntax zu Achten, ein vorangestelltes „normalize“ als Befehl, damit das Programm die genannten Konstante auch als solche zu erkenne vermag.

Um sich über die beiden im Beispielabschnitt verbleibenden Enum Variablen zu informieren, wird der Nutzer ermutigt sich mit dem dafür eigenen JavaDoc Eintrag zu beschäftigen. Hierdurch wird der Anwender zugleich an diese Dokumentation herangeführt, welche ebenso für die Appgenerierung von Bedeutung ist, da diese die maschinell erzeugte Funktionssammlung von Java selbst ist.

Das Unterthema der „String function“ stellt abschließend lediglich einen programmiertechnischen Kniff dar und präsentiert die Möglichkeit den Datentyp String in der Appprogrammierung zu nutzen. Ein Einsatzgebiet hierfür ist die Layoutanpassung, auf welche im Kapitel 4.5 eingegangen wird.

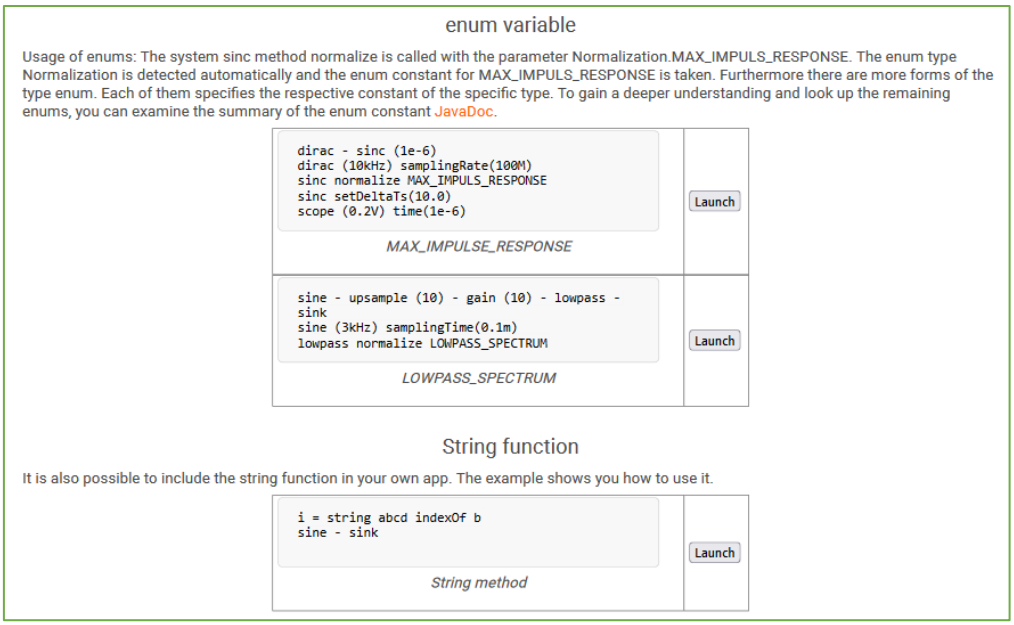


Abbildung 24 "Enum variable und String function" -myExample

4.4 MyLabAlive – Videoerstellung für Beispielabschnitt

4.4.1 Vorbereitungen und Planung für das Tutorialvideo

An das im Kapitel 4.3 angekündigte Video wurden folgende Voraussetzungen gesetzt. Zur Erinnerung, dieses soll den automatischen Prozess der Programmcodeerstellung für die vom Nutzer getroffenen internen Appeinstellungen, Status und Positionen der jeweiligen Appprogrammfenster zeigen:
Im Wesentlichen sollte dem Nutzer die Handhabung der Seitenoberfläche, sowie die Handhabung der ausgeführten App, insbesondere deren Speicherfunktion für die Übernahme der getätigten Einstellungen vorgeführt werden.
Zusätzlich zu dieser visuellen Präsentation war es im Voraus auch wünschenswert, dass das endgültige Video mit einer klaren Beschreibung durch gesprochenen Text begleitet wird, der die im Video gezeigten Arbeitsschritte kommentiert.

Bei der Entwicklung des Videos ergab sich obendrein noch folgende Gelegenheit:
Die Anleitungsversion my2 umfasste in ihrer Fassung eine eigene Unterseite als Anleitungsbereich, die sich der für den Nutzer notwendigen Accounterstellung und Speicherung des generierten Signals widmete („Safe My Work“). Jedoch nahm diese auf eine vergangene Version dieses Prozesses Bezug, der so in der aktuellen myLabAlive-Version nicht mehr existierte. Denn für die zukünftige unkomplizierte Verwaltung der selbstgebauten Signale wurde durch die Arbeit der Bacheloranden die Seite in Ihrer Funktionalität und Aufmachung überarbeitet und unteranderem zur

Signalspeicherung ein eigens hierfür intuitiver Knopf entwickelt, der sich unmittelbar bei der Text2App Umgebung, also direkt bei der Appgenerierung, befindet.
Durch diese Vereinfachung bot es sich an, die Anleitung von dieser zusätzlichen Unterseite zu befreien und die neue Funktionalität mit diesem Video abzudecken.
Besagtes Video wird mit der in Abbildung 25 gezeigte Benutzeroberfläche eingeleitet. Hier agiert der Nutzer, um seine gewünschte App zu erstellen und ihre Speicherung durchzuführen. Im Folgenden findet sich ihre Erläuterung, wie sie auch im Video stattfindet:

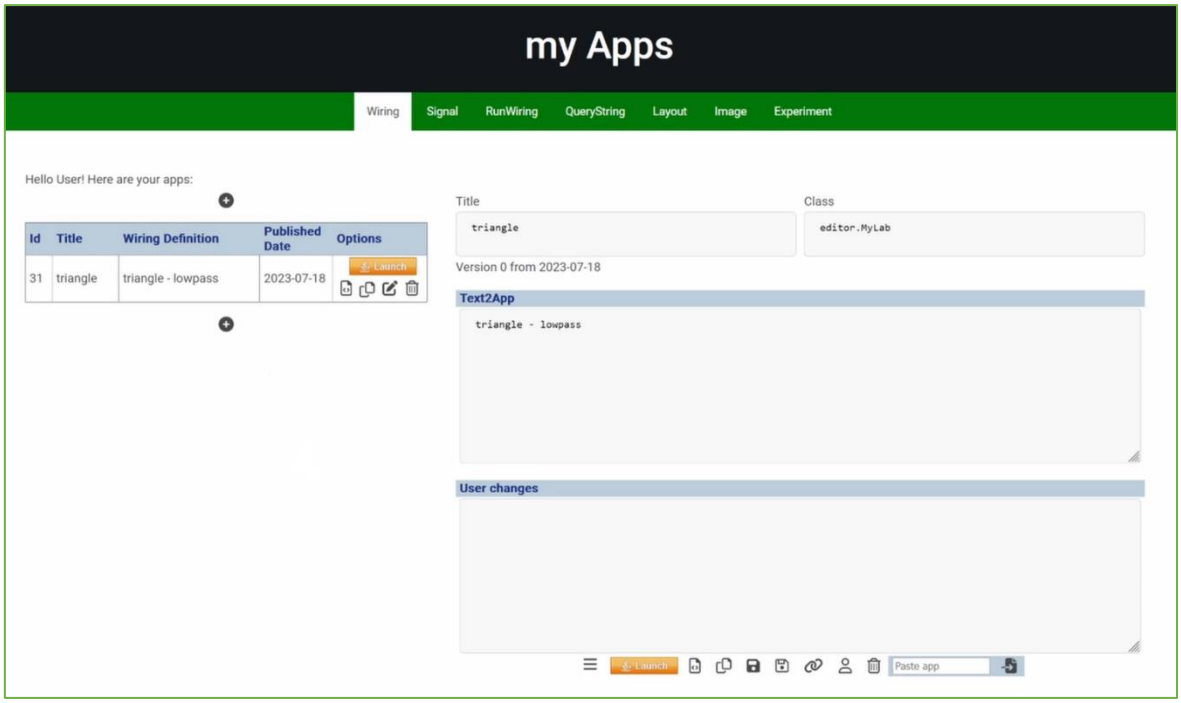


Abbildung 25 Nutzeroberfläche des MyLabAlive´s

Auf der linken Seite der gezeigten Oberfläche findet die Auflistung und Verwaltung aller vom Nutzer bislang erstellten Wirings/Apps statt. In dem Video wird hier zu Demonstrationszwecken ein Wiring in Form einer einfachen App unter dem hierfür angelegten LabAlive-Account „User“ erstellt. Auch bei mehreren gelisteten Wirings wäre diese durch ihren personalisierten Titel und das dokumentierte Erstellungsdatum eindeutig. Durch Betätigen des Editieren-Buttons (Stift/Schreibsymbol) kann die jeweilige App selektiert und an dieser weitergearbeitet werden.
Auf der rechten Seite befindet sich die Oberfläche zur Appgenerierung. Bei klassischer Nutzung, wie sie auch im Video präsentiert wird, wird hierbei mit den obigen Textfeldern begonnen. Hier kann der Nutzer seiner zukünftigen App einen personalisierten Titel vergeben, das Textfeld der Klasse ist vordefiniert und benötigt keiner weiteren Veränderung. Im „Text2App“-Textfeld darunter findet die eigentliche Appprogrammierung statt.
Exemplarisch wird hier ein Dreieckssignal erstellt, dass auf einen Tiefpassfilter trifft. Da selbst keine weiteren Parameter vordefiniert werden müssen, werden auf beiden Systemen die Default-Settings

angewandt, hierzu später mehr. Getreu der Anleitung ist es nicht notwendig als Signalabschluss die Sinke extra zu erwähnen, hier wurde sie ebenso eingespart.

Das letzte Textfeld „User changes“ ist jenes, auf dass das Video zur Funktionalitätsvorstellung abzielt. Denn in diesem lassen sich die vom Nutzer gespeicherten Einstellungen der App als automatisch generierter und fertiger Programmcode wiederfinden, nach dem diese in der laufenden App nach den eigenen Vorstellungen interaktiv getätigt und festgehalten wurden.

Am unteren Rand des „User changes“ Textfeldes finden sich bei aktuellem Stand abschließend noch die Buttons, erstellt durch die Bacheloranden, von denen manche bereits in Kapitel 4.3 kurz angesprochen wurden.

Von links nach rechts erklärt umfassen diese wie in Abb. 26 folgende Funktionalität:

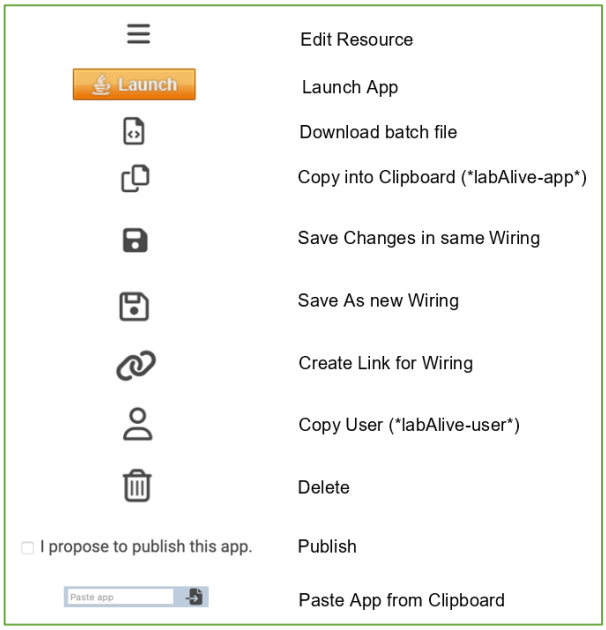


Abbildung 26 Buttonerläuterung - myLabAlive

Da an der Oberfläche myLabAlives jedoch auch in Zukunft weitergearbeitet wird und sich diese weiterentwickeln und abändern kann, wird im Video nur auf die essenziellen eingegangen. Diese umfassen das Speichern und das Launchen der App.

Beides war zu Beginn jedoch mit Problemen versehen, da die diesjährigen Bacheloranden aufgrund der Studienstruktur und der früheren Bachelorarbeitsdeadline diese Funktionalitäten nicht mehr ausreichend und fehlerfrei in das bestehende LabAlive Projekt integrieren konnten.

Aufgrund der fortgeschrittenen Zeit im eigenen Projekt fiel der zeitliche Spielraum jedoch zu knapp aus, als dass sich die eigene Arbeit dadurch hätte lange verzögern dürfen, noch dass sie sich zusätzlich der verpassten Codereparatur der Arbeiten aus anderer Quelle hätte antuen können.

In Rücksprache mit Prof. Dr. -Ing. Erwin Riederer und dem Mitarbeiter Stabsfeldwebel Jürgen Unfall wurde deshalb beschlossen, dass sie die Code-Kinderkrankheiten der Buttonintegration und der Speicherfunktion innerhalb der Java App nachgehen, damit sich diese Arbeit weiterhin rein auf die unter Kapitel 1.3 definierte Aufgabenstellung konzentrieren und diese ungehindert erfolgreich zum Abschluss bringen kann.

Anschließend an diese essenziell gehaltene myLabAlive Oberflächenerläuterung soll sich in weiterer Planung dann die ausführende App anreihen. In dieser soll dem Nutzer etwaige Appanpassungen gezeigt werden, wie sie interaktiv leicht erbracht werden können. Nach dieser erfolgten Anpassung soll dann die Speicherfunktion der App beleuchtet werden. Also wie und was die Speicherfunktion abdeckt, sowie wo der dann maschinell automatisch erzeugte Programmcode vom System festgehalten wird, damit in zukünftigen App-Launches die zuvor gewünschten interaktive erbrachten Appanpassungen bequem mitgeladen werden und als Initialisierung dienen können.
Zur Veranschaulichung soll am Ende die bearbeitete App, mit dem maschinell vom System erbrachten „User changes“, erneut über die myLabAlive Browseroberfläche gestartet werden, damit sich der Nutzer selbst vom Ergebnis einen Eindruck schaffen kann und der Prozess damit untermalt wird.
Hiermit lässt sich das Video in drei Szenen unterteilen. Szene Eins umfasst die einleitenden Worte in das Thema des Videos, sowie der Appeinrichtung auf der myLabAlive Browseroberfläche. In Szene Zwei wird diese App dann aufgegriffen, zur Präsentation für den Nutzer in ihren Parametern und Darstellung geändert und nach erfolgter interaktiver Anpassung gespeichert. Szene Drei bildet den Schluss und zeigt den maschinell erzeugten Programmcode dieser präsentierten Veränderungen, wie sie in der App gespeichert wurden und führt das finale Ergebnis vor.

4.4.2 Videoaufnahme

Anfangs wurde versucht, das interne Aufnahmeprogramm von Windows über die Xbox Game Bar zu verwenden. Dieses versprach eine einfache Aufnahme, ohne auf Drittanbieter-Programme zurückgreifen zu müssen. Jedoch wurde schnell klar, dass das Programm für diese Aufgabe nicht ausreichend ist. So ließ sich hierrüber lediglich eine zuvor ausgewählte Anwendung und nicht der komplette Bildschirm aufnehmen. Für Szene Eins und zumindest in Teilen Szene Drei wäre dies noch verschmerzbar gewesen, da hier als Anwendung der Browser selektiert werden und damit alles vom Aufnahmeprogramm erkannt und eingefangen hätte werden können. Szene Zwei schloss dieses Aufnahmeprogramm jedoch in Gänze aus, da die Java Simulationsapp in mehrere Programmfenster unterteilt ist und damit eine Begrenzung auf ein einzelnes keinen Sinn ergeben hätte. Zumal für die weitere Videobearbeitung ein im Hintergrund zu den App-Programmfenstern gehaltener Greenscreen eingestellt sein musste, um diesen für die optische Szenenabstimmung im weiteren Videoschnitt wegkeyen(=ersetzen) zu können. Somit ist dieses Programm in Szene Zwei selbst sogar am notwendigen Hintergrund gescheitert, da es diesen als Anwendung niemals verbunden mit einem weiteren Programmfenster hätte aufnehmen können.

Nach weiterem recherchieren und experimentieren fiel die Wahl dann auf das Drittanbieter-Programm OBS Studio, das vor allem aus der Streaming-Szene bekannt ist (Programmvorstellung siehe Kapitel 2.3).

Dieses verlangte ein paar Versuche, um die idealen Programmeinstellungen zu finden, wie das Abstimmen der Aufnahmequalität (FPS-„Frames per Second“ =Bildwiederholrate, Bildschirmauflösung) und des Videoencoders mit der erbringbaren Leistung der zur Verfügung stehenden Hardware. Nach ein paar Versuchen und unter Beobachtung der Hardwareleistungsgrenze wurde sich für eine Bildwiederholrate von 30Hz (=30FPS) und dem Videoencoder „Software (x264)“ entschieden. Diese Bildwiederholrate ist vertraut und wird auch für gängige Kinofilme verwendet. Alternativ stand für letzteres auch ein auf der Hardware basierende Encoderlösung „NVENC, H.264“ und „NVENC, HEVC“ bereit, jedoch nahmen diese zu viel Arbeitsspeicher und CPU-Leistung in Anspruch.

Des Weiteren bot das Programm die Option der Hochskalierung der Bildschirmauflösung, wobei dies jedoch mit grafischen Artefakten, Bildaussetzern und einer höheren CPU-Auslastung einherging und den ohnehin schon mit Bedacht gewählten Software Encoder in Verbindung mit der verfügbaren Hardware an seine Grenzen brachte. Dennoch wurde eine höhere Auflösung als die bisherigen auf LabAlive existierenden Tutorialvideos in Full-HD (1920x1080 =2.073.600 Pixel) als angemessen für die heutzutage gängige Bildschirmtechnik erachtet. So wurde kurzerhand die verwendete Hardware über ein HDMI-Kabel um einen WQHD-Monitor (2560x1440 =3.686.400 Pixel) erweitert, auf diesen die zu tätigende Aufnahme nun nativ mit einer fast doppelt so hohen Bildschirmauflösung erfolgen konnte wie zuvor, ohne den Softwareencoder zu stark über seine Grenzen zu bringen.

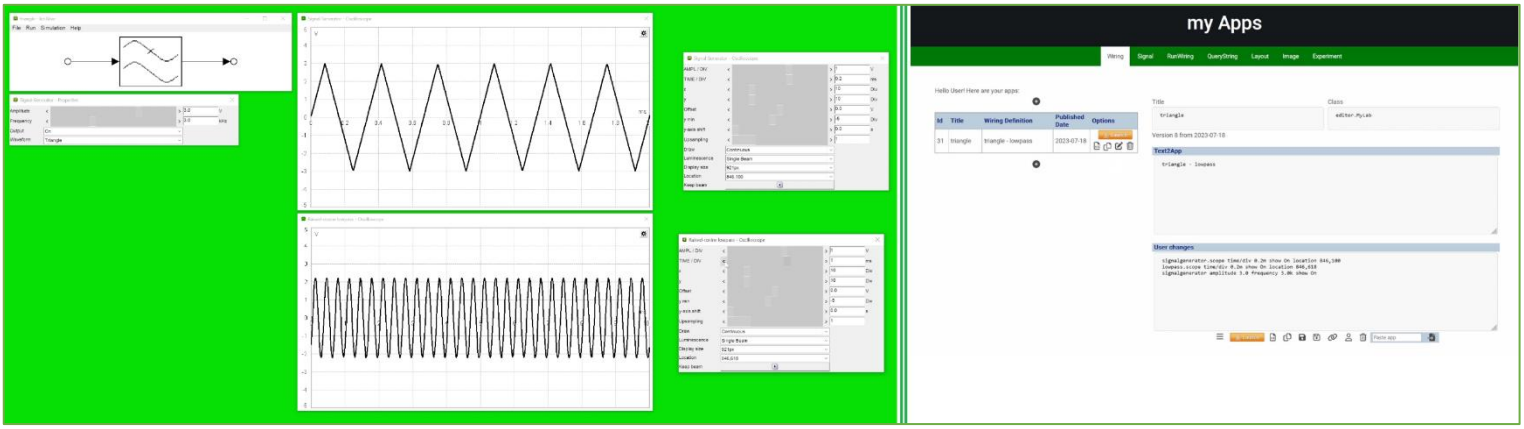


Abbildung 27 Szenenaufnahme -Tutorialvideo myLabAlive

Abbildung 27 zeigt Ausschnitte aus den gefertigten Aufnahmen. Die linke Hälfte von Abbildung 27 stammt aus Szene Zwei, in der die interaktive Appanpassung vollzogen wurde. Für das aufzunehmende Video umfasste dies die Anpassung der Parameter des Eingangs-Dreieckssignals, da dieses wie zuvor beschrieben nur mit den Default-Settings generiert wurde. Präziser ausgedrückt

wurde hier der Amplitudenparamter von 1V und der Frequenzparamter von 1kHz auf 3V und 3kHz gesetzt.
Des Weiteren wurden die Anzeigen beider Oszilloskope insoweit manipuliert, als dass sie ihr jeweiliges Signal optisch passender einfangen. Für beide umfasste dies eine Verringerung des „Time per Division“-Parameters in den Anzeigeneinstellungen.
Getreu dem verfolgten Hauptziel des Videos wurden diese parametrischen Anpassungen und Status der geöffneten Appfenster dann über die Appinterne Speicherfunktion festgehalten und mit einem Übergang zu Szene Drei, zurück auf der Browseroberfläche, im „User changes“-Textfeld präsentiert. Ein Szenenausschnitt samt der fertigen und vom System automatisch maschinell erzeugten Programmierbefehle findet sich in der rechten Hälfte von Abbildung 27.

Nach Aufnahme aller Szenen galt es noch, den audiovisuellen Anteil, also die videobegleitenden Kommentare zu erzeugen. Dies wird im Folgekapitel behandelt.

4.4.3 Erstellung der Kommentarbegleitung des Videos

Durch die Inspiration von bereits getätigten Audioaufnahmen für LabAlive aus ehemalig durchgeführten Arbeiten (Signalgenerierung im Rahmen der Projekt- und Bachelorarbeit) wurde entschieden, hierfür eine Sprachmaschine einzusetzen, die den selbst geschriebenen Text vorliest und diesen damit selbst audiovisuell verarbeitet. Genutzt wurde hierfür die Sprachmaschine von https://www.narakeet.com/. Hierbei handelt es sich eigentlich um ein kostenpflichtiges Produkt, jedoch erlaubt es zu Demozwecken bis zu 20 kostenlose Verarbeitungen, welche gerade so ausreichten. Narakeet bietet hierbei die Wahl zwischen, allein in der englischen Sprache, elf unterschiedlichen Dialekten, für die eigenen Zwecke wurde sich hierbei für einen leicht verständlichen und möglichst natürlich klingenden Sprecher mit amerikanischem Dialekt entschieden. Als weitere vorweg einzustellende Parameter wurde das finale Dateiformat auf .mp3 festgelegt und nach ersten Versuchen letztlich das Audiopegelvolumen sicherheitshalber auf „loud“ umgestellt.
Damit in der späteren Videobearbeitung diese .mp3-Dateien flexibler eingefügt werden können, wurde bis auf wenige Ausnahmen jeder Satz separat vertont. Dies räumt mehr Freiheiten ein, zum Beispiel, dass das Bildmaterial einfacher nach den vertonten Kommentaren ausgelegt werden kann und nicht andersherum. Ansonsten hätte anderenfalls die Audiospuren für die gewünschte Bild-/Audiokombination oftmals gespult, gezerrt oder geschnitten werden müssen, mit dem Risiko, dass die Audiospur undeutlich wird oder gar verformt. Dies wird noch im Folgekapitel „4.4.4 Videoverarbeitung, -schnitt und Rendering“ deutlich.

Abschließend für dieses Kapitel findet sich zu Dokumentationszwecken die entwickelte englischsprachige Videobegleitung, ein jeder Absatz verkörpert eine separate Audiodatei:

Szene 1:

In this tutorial, you will learn that generating your own app does not require endless hardcoding. Instead, many parameter settings can be made interactively in the app, as long as at least the basic structure of your circuit is defined.

As a quick example, a simple circuit is built in which a triangle signal meets a low-pass filter. This description is already sufficient for our app.

Don't forget to save your app description so that you can still access and manage it later.

The app can be downloaded and opened via the launch button, this may take a short moment.

Szene 2:

To make a deeper app adjustment, we open the displays of our virtual oscilloscopes of the circuit, as well as the properties window of our triangle signal generator.

Here we can see that the previous input signal is predefined by default with an amplitude of one volt and a frequency of one kilohertz. We are now free to adjust this as we wish. As an example, we can now set an amplitude of three volts and a frequency of three kilohertz.

The next adjustment we make is purely visual. We can adjust the display window of the oscilloscopes to show our respective signal more clearly. In this example, it is sufficient to lower the time per division parameter. This step is repeated for both oscilloscopes.

It is emphasized that the position and status of each display window will be captured. Thus, in this example, we intentionally leave both oscilloscope displays and the signal generator properties window open for future launches.

Now that all the desired settings have been made, we save the data of our app and the system automatically creates the necessary programming commands for us.

Szene 3:

If we now switch back to our myLabAlive browser interface and reload our app, all settings will have been automatically generated in the user changes window, as we had previously saved them in the app.

With each new launch, these settings will now be taken into account and the app will appear as we intended.

And there you have it, you have now successfully set up your own app interactively without having to enter any further hardcoded programming commands yourself.

4.4.4 Videoverarbeitung, -schnitt und Rendering

Um die Videodateien (.mp4) mit den Kommentaraudiodateien (.mp3) zu vereinen, wurde auf das Schnittprogramm „DaVinci Resolve“ von „Blackmagic Design“ gesetzt (Programmvorstellung siehe Kapitel 2.4).

Trotz fehlender Vorerfahrung unterstützte die Anwendung zumindest über die didaktisch in der richtigen Reihenfolge sortierten Programmabschnitte. So wird über den Menüpunkt „Media“ zuerst sämtliches Material, sowohl Video- als auch Audiodateien, eingespielt und dann unter dem Programmabschnitt „Edit“ via Drag and Drop in einer sogenannten Timeline eingebunden und mit einer Vielzahl an Werkzeugen angepasst.

Ein Ausschnitt aus dem Editierungsprozess findet sich in Abbildung 28.

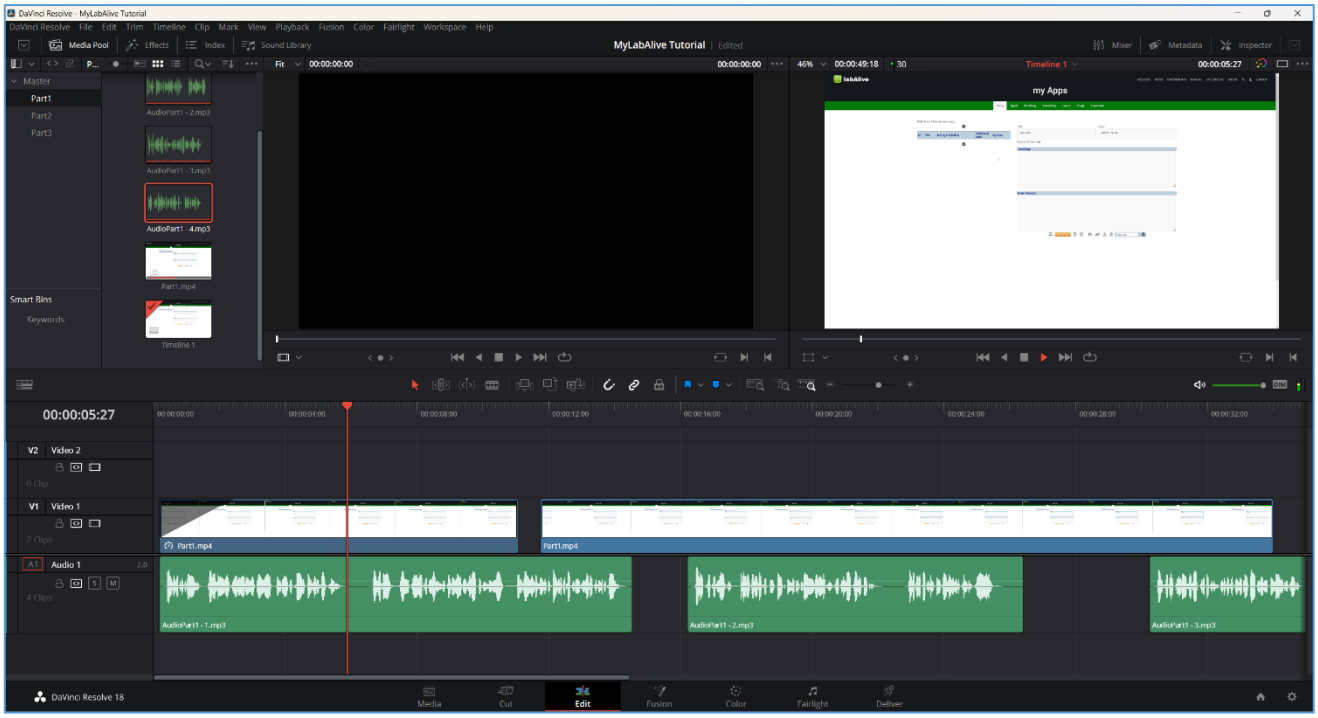


Abbildung 28 DaVinci Resolve - Editierungsprozess

Dieser stellt sich wie folgt dar: In der oberen linken Ecke der Abbildung 28 ist der Überblick über das jeweilige eingefügte Material zu sehen. Zur besseren Übersichtlichkeit wurde dies bereits nach den beschriebenen Szenen aus Kapitel „4.4.1 Vorbereitungen und Planung für das Tutorialvideo“ vorsortiert und dessen jeweilige Audiodateien mit ihrer Reihenfolge beziffert.

In der unteren Hälfte der Abbildung findet sich hingegen die benannte Timeline. Diese ist in Video- und Audiobereich unterteilt und dort findet der Großteil der Bearbeitung statt. Hier wird das Material nicht nur geschnitten und zusammengefügt, sondern auch Übergänge eingesetzt oder auch Videos in ihrer Geschwindigkeit angepasst. So wurden beispielsweise für die Verständlichkeit des Videos die einleitenden Worte und damit die Tonspur mit einem eingefrorenen Standbild begleitet,

damit alle Infos geordnet nacheinander erfolgen können und das Video nicht zu weit voranschreitet. Dies ist beispielsweise an dem beigefügten Geschwindigkeitssymbol des ersten blauen Videobausteins in der Timeline zu sehen.
Generell kann festgehalten werden, dass im gesamten Videoprojekt durchgängig auf diesen Videoschnitttrick zurückgegriffen wurde. Dabei wurde entweder einzelne Bilder herausgeschnitten und situative eingefroren oder die Geschwindigkeit einzelner Videoclipanteile angepasst, um eine optimale Synchronisation zwischen Bild und Tonmaterial zu erreichen. Zusätzlich hierzu wurde auch auf diesen Videoschnitttrick gesetzt, um lang ladende Prozesse innerhalb des Videos entweder gänzlich rauszuschneiden oder zu beschleunigen, um über die in Kapitel „4.4.2 Videoaufnahme“ beschriebene überschaubare Hardwareleistung des Aufnahmegerätes hinwegzutäuschen.
Die in ihrer Mehrheit in einzelne Sätze separierten Audiodateien boten für die Synchronisation den benötigten Freiraum, um das Videomaterial fein genug auf die jeweiligen Tonspuren abzustimmen. Diese Arbeitstechnik beschreibt die hauptsächlich zu tätigenden Handlungsschritte der Szene Eins und Drei.

Wie im vorherigen beschrieben, wurde in Szene Zwei mithilfe eines sogenannten „Greenscreens“ separiert die Arbeiten mit der ausgeführten App eingefangen. Um diesen Greenscreen für das Video nun entfernen und ersetzen zu können, musste der Programmabschnitt „Fusion“ von DaVinci Resolve genutzt werden. Dieser ist in Abbildung 29 dargestellt:

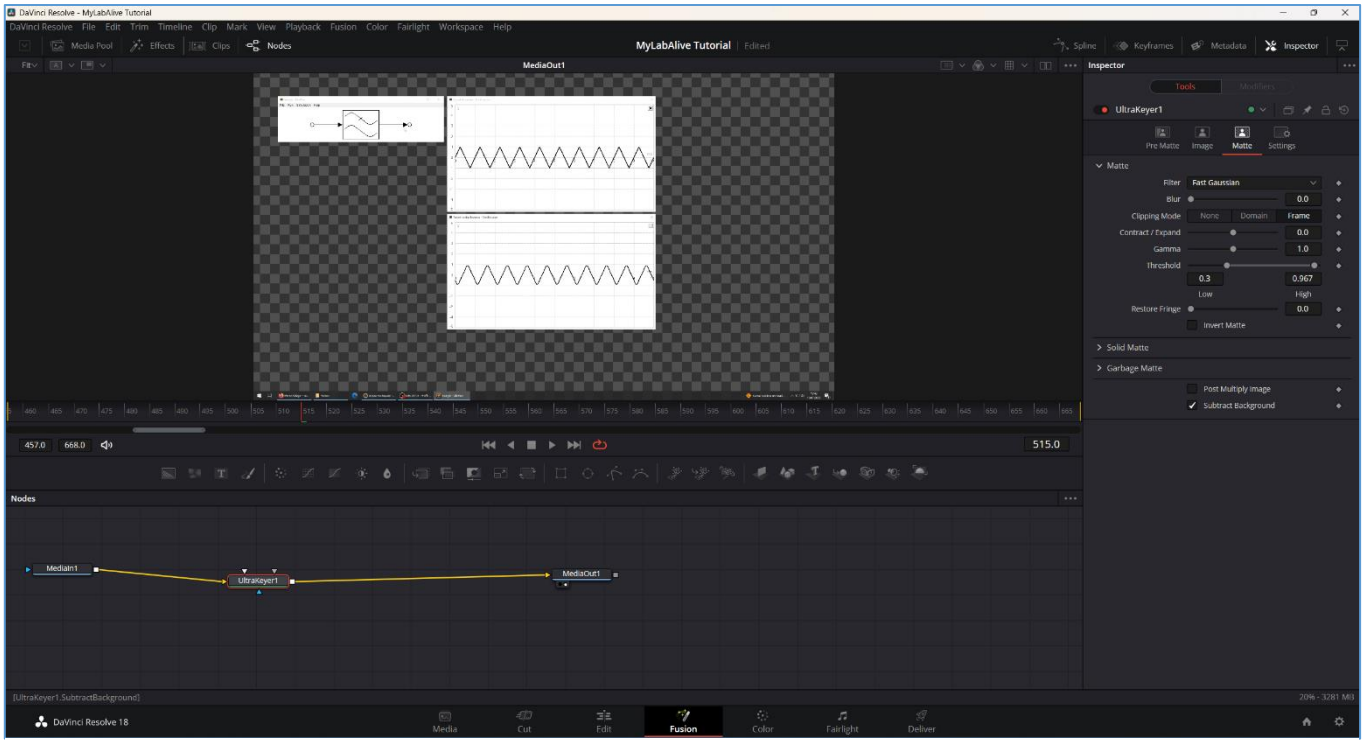


Abbildung 29 DaVince Resolve - Fusionprozess für Greenscreen

Als großer Unterschied zur Oberfläche in Abbildung 28 steht das sogenannte „Node“-Fenster in der unteren Hälfte der Anzeige. In diesem wird auf grafischer Ebene per Drag and Drop Funktionsblöcke eingebunden und miteinander verlinkt. Um wie beabsichtigt den Greenscreen zu entfernen, wurde hier auf den Funktionsblock „UltraKeyer“ gesetzt, dessen herauszukeyende Farbe mittels virtueller Farbpipette frei definierbar ist und der sich zwischen „MediaIn1“ und „MediaOut1“ zu positionieren hat. Die Verlinkung ist so zu verstehen, dass nach finalem Fusion-Prozess das Datenmaterial des MediaOut bildgebend für das Video ist. MediaIn hingegen wirkt nur als Quelle für den Bearbeitungsprozess und kann in ihrer Verlinkung bis MediaOut nach Belieben unterbrochen und bearbeitet werden.
Der Baustein des „UltraKeyers“ bot die gesuchte Funktionalität, musste jedoch für seine erzielbare Genauigkeit in seinen Parametern verfeinert werden. Präziser ausgedrückt, musste der Parameter „Threshold“ angepasst werden. Dieser ist in Abbildung 29 auf der rechten Seite im Einstellungsfenster des „UltraKeyers“ aufgeführt und wirkt sich auf die Kantengenauigkeit bei der Hintergrundentfernung aus.

Die komplette Szene wurde aufgrund fehlender Vorerfahrung und aufgrund des inspirierenden branchendominierenden Einsatzes mittels eines Greenscreens verwirklicht. Im Verlauf der Videobearbeitung stellte sich jedoch heraus, dass die Farbe des Greenscreens von Icon-Elementen aus der ausgeführten LabAlive-App geteilt wurde. Dies stellte insoweit ein Problem dar, als dass nun jene LabAlive-Icons ebenso von der Funktionalität des „UltraKeyers“ betroffen waren und damit fälschlicherweise transparent wurden. Dies hätte mit Vorwissen und der frei wählbaren Farbe für den UltraKeyer und über die Wahl eines anderen Farb-Screens für die Szene, leicht vermieden werden können, musste nun jedoch berichtigt werden.

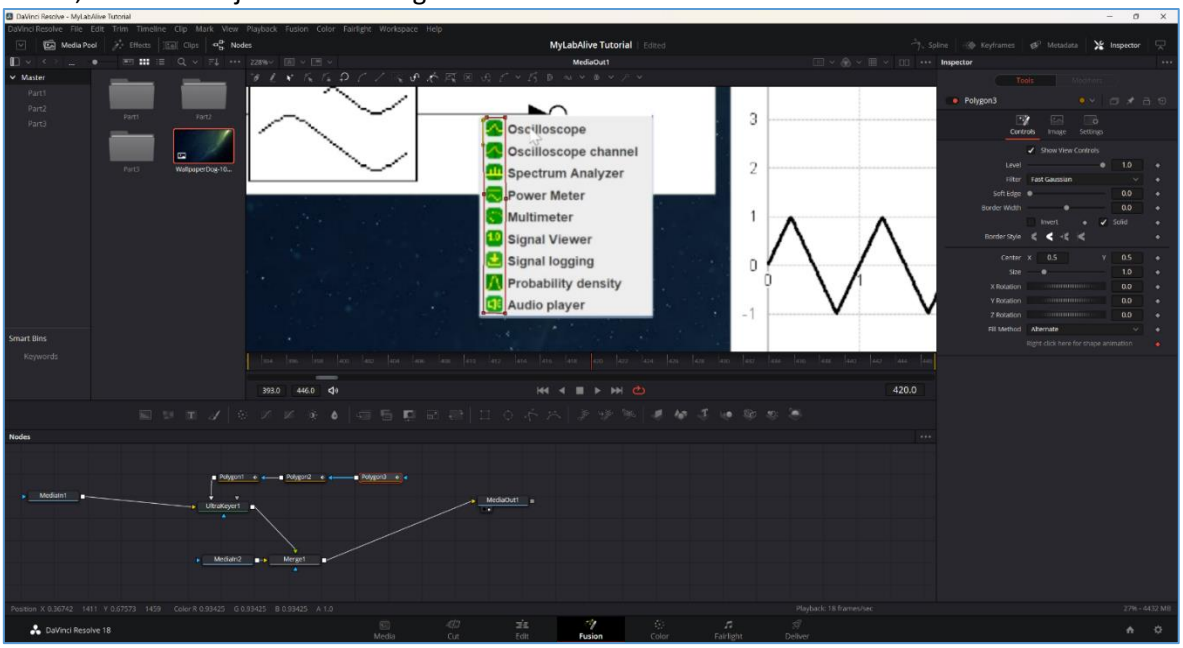


Abbildung 30 DaVinci Resolve - Greenscreen Problemlösung mittels Polygone

Abhilfe schaffte, wie in Abbildung 30 gezeigt, extra zu diesem Zweck eingefügte Polygone. Diese wurden mit in den Bearbeitungsprozesses des Fusions eingefügt, um dem UltraKeyer in ihren aufgespannten Bereich eine Ausnahmezone aufzuzeigen.
Hierfür wurden die Polygone als solide beziehungsweise „unbetroffene“ Maske dem UltraKeyer übergeben. Die Problemlösung war damit recht simple, jedoch auch sehr aufwendig, da diese Ausnahmezonen gesamtheitlich für jeden einzelnen Videoausschnitt galten. Dadurch mussten die Videobausteine der Timeline in abhängig der situativ zu setzenden/zu entfernenden Polygone extra aufgetrennt und geschnitten werden, um eine feine Anpassung der Ausnahmezonen über die einzelnen Videobausteine zu ermöglichen. Bei genauer Betrachtung kann man diese angewandte Problemlösung auch noch in den einzelnen Appfenstern des Endresultats erahnen, da die Ausnahmezonen nicht beliebig Bildratengenau gesetzt werden konnten und damit die betroffenen Icons kurz vor der Ein- beziehungsweise Ausblendung transparent erscheinen. Dieses Nachwirken ist jedoch auf ein absolutes Minimum reduziert.
Der nach dem UltraKeyer folgende Merge-Baustein im Node-Fenster der Abbildung 30 führt hingegen ein zweites Bildmaterial hinzu. Bei diesem handelt es sich um den Hintergrundersatz. Für diesen wurde ein dunkles Bild ausgewählt, damit vor allem die ausgeführte App hervorsticht und des Nutzers Aufmerksamkeit einfängt. Zugleich wurde jedoch für das Ersatzhintergrundbild auch auf einen in den Farbton Grün übergehenden Farbverlauf geachtet, um einen geschmeidigeren Szenenübergang von der LabAlive Oberfläche heraus-/herein zu ermöglichen, da diese wiederum selbst grüne Farbakzente besitzt.
Das fertige Fusion-Ergebnis ist exemplarisch als Szenenausschnitt in Abbildung 31 gezeigt. Damit die vorherige Taskleiste der Szene Zwei am unteren Bildschirmrand verschwindet, wurde das Video leicht gezoomt und in Gänze nach unten hinweggeschoben.

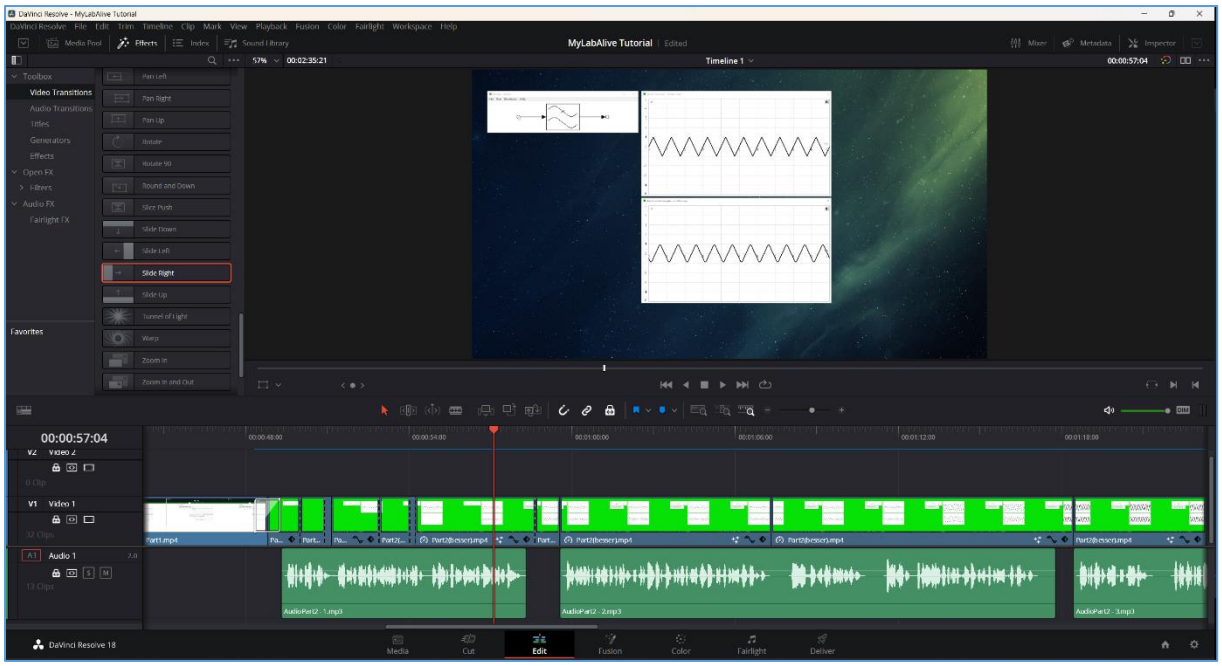


Abbildung 31 DaVinci Resolve - finales Ergebnis nach vollständigem Fusion-Prozess

Nachdem alle Videoelemente der Szenen bearbeitet und zufriedenstellend mittels Szenenübergängen verbunden und zugleich sämtliche Tonspuren von der Audioart „Mono“ zu „Stereo“ gewechselt wurden, konnte das Videoprojekt als Renderauftrag abgeschlossen werden.

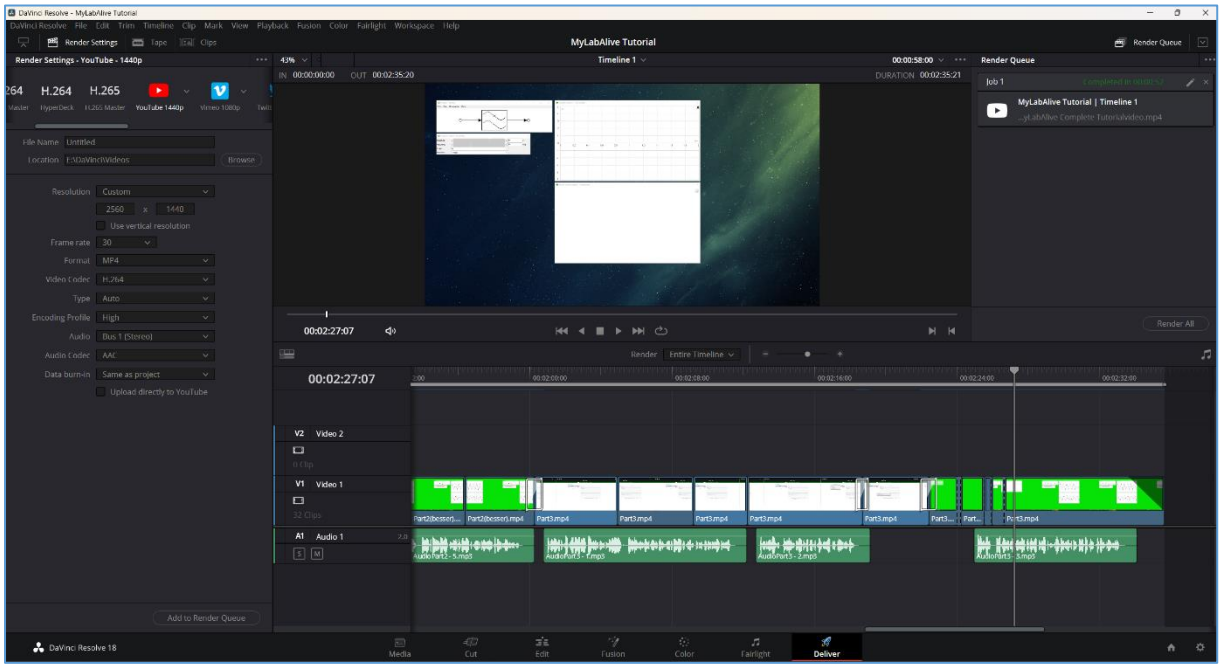


Abbildung 32 DaVinci Resolve – Render-Einstellungen

Beim abschließenden Prozess des Renderns wurde damit erneut, wie in Abbildung 32 gezeigt, die zu erzielende Frame Rate und Auflösung, sowie das Dateiformat (.mp4) und der Videoencoder „H.264“ eingestellt. Das finale Video wird das Herzstück der neuen Anleitung zu myLabAlive und stellt hiermit auch zugleich das zuletzt zu erklärende Element dessen myExample Unterseite dar.

4.5 MyLabAlive – Überarbeitung und Einführung der Layout Befehle „Layout Creation“

Wie in Kapitel 4.3 als eine der letzten „myExample“-Funktionsoberthemen angekündigt, wurde für das Thema der Layoutanpassungen der App eine eigene Unterseite für die Anleitung generiert. In dieser wird dem Nutzer präsentiert, wie zur weiteren App-Kontrolle und Gestaltungsfreiheit die eigenen Schaltungen in ihrer Systemanordnung angepasst werden kann. Hierzu muss der Nutzer seine Systeme mit Richtungsangaben versehen. Eine Anleitung hierzu bestand bereits in my2, doch die Erklärungstexte geboten einer Nachbesserung und durch die diesjährigen Arbeiten der Bacheloranden kam eine zusätzliche detaillierte Layoutanpassung hinzu, die bisher noch nicht thematisiert wurde.

Der Aufbau der zu überarbeitenden Seite stellte sich wie folgt dar:

Zu Beginn werden beide zu Verfügung stehenden Programmierstile der Layoutanpassung beschrieben, voneinander differenziert und zugleich die zu verwendenden Layoutkommandos eingeführt.

Für deren Präsentation finden sich im Folgenden drei Showcase-Beispiele, in denen diese genutzt werden. Und zum Abschluss wird dem Nutzer eine Übungsaufgabe gestellt, die zum Nachahmen und Einstudieren einlädt.

Als ersten zu überarbeitenden Punkt wurden die Beschreibungstexte angepasst. So wäre der Nutzer beispielsweise bisher mit einer unpassenden direkten Ansprache zu dem hier behandelnden Abschnitt willkommen geheißen worden. Stattdessen wurde das Intro verschlankt und der Inhalt pragmatischer beschrieben. Vorwegnehmend dessen, was im weiteren Verlauf dieses Kapitels folgt, wurde auch die Tabelle der Layout-Kommandos mit der intentionaleren Reihenfolge der Tabelle der Halbschritt-Kommandos vereinheitlicht. Einen vorher/nachher Vergleich findet sich in Abbildung 33:

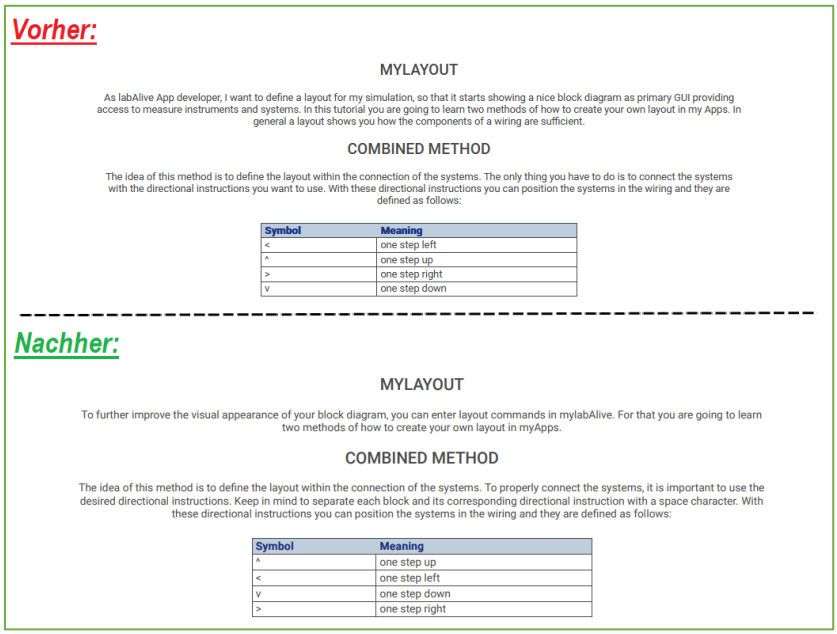


Abbildung 33 myLabAlive - Layout Creation – Intro

Des Weiteren fielen in der vorherigen my2-Anleitungsversion störende Formatierungsfehler der Showcase-Programmierzeile auf, die verbunden mit der Textüberarbeitung ebenso gelöst wurden.

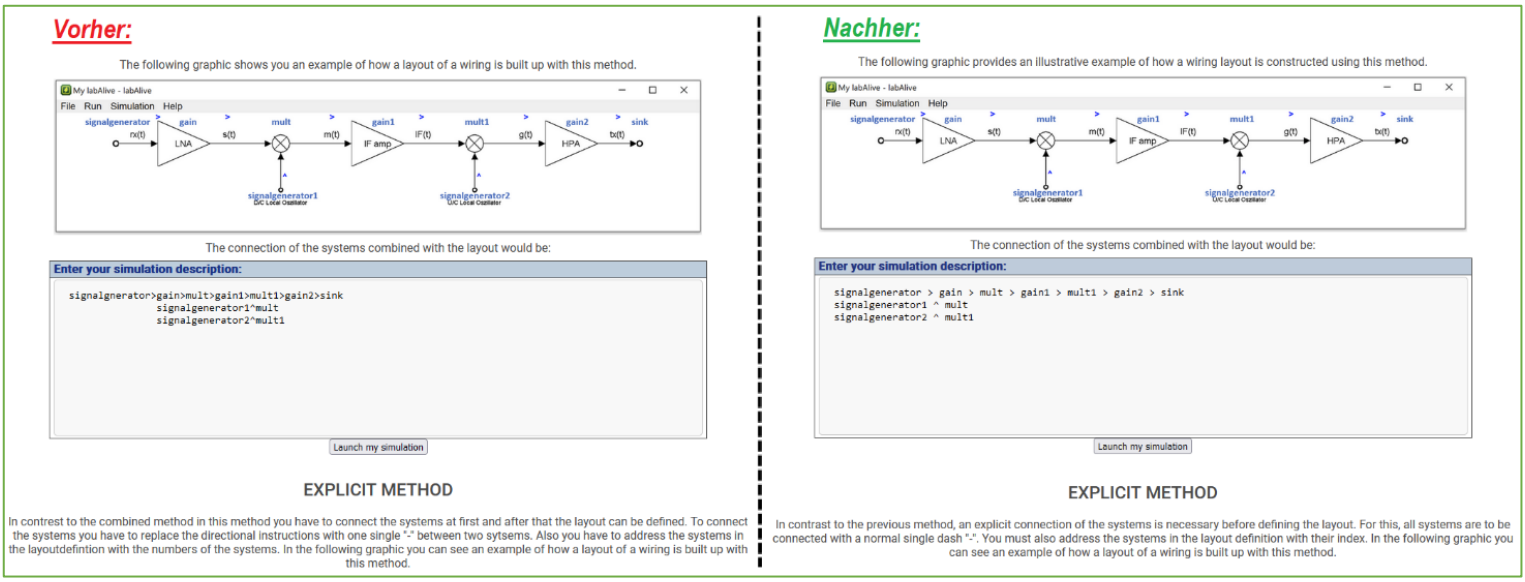


Abbildung 34 myLabAlive - Layout Creation – Intro

Die neuen detaillierten Layoutbefehle dienen als zusätzliche und feineren Anpassungen und ermöglichen in grafischen Halbschritten die gewünschten Systemblöcke zu platzieren. Damit der Beginn der Seite jedoch nicht mit Layout-Kommandos überladen wirkt und die neu eingebrachten als klare rein optionale Erweiterung präsentiert werden, wurde deren Einführung nach den Showcase-Beispielen platziert. Also im letzten Drittel der Unterseite. Deren Darstellung findet sich in Abbildung 35:

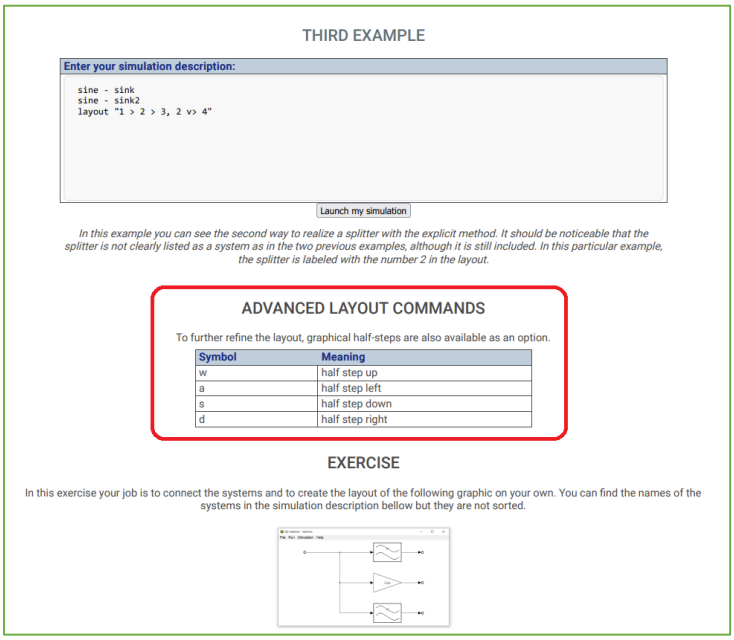


Abbildung 35 myLabAlive - Layout Creation - Advanced Layout Commands

Aufgrund der erfolgten Nachbesserung der Unterseite „Layout Creation“ und dem in Kapitel „4.4.1 Vorbereitungen und Planung für das Tutorialvideo“ beschriebenen Gelegenheit, die ehemalige my2-Anleitungsunterseite „Safe My Work“ einzusparen, wurde die Unterseite der „Layout Creation“ in ihren Anleitungsreihenfolge nach vorne gesetzt und findet sich in der neuen Version nun direkt hinter der Beispielseite „myExample“, statt wie zuvor erst auf der hierzu drittfolgenden Unterseite. Hierdurch wird sich ein didaktisch zusammenhängender Lernprozess des Nutzers versprochen.

4.6 Überarbeitung JavaDoc Index

4.6.1 Ausgangslage und Problemstellung

Im Verlauf dieser Masterarbeit wurden bereits mehrmals Verweise und Verlinkungen auf **JavaDoc** angesprochen. Bei diesem handelt es sich um eine aus dem Quelltext des Projektes maschinell erzeugte Dokumentation von Java selbst.

Zusätzlich zu dieser Dokumentation existierte jedoch seit der überarbeiteten Anleitungsversion "my2" eine vereinfachte JavaDoc-Variante namens **JavaDoc Index**, die dem Nutzer bereits im myLabAlive-Abschnitt selbst die grundlegenden Systeme auf einer eigenen Unterseite der Anleitung vorstellte.

In damaliger Besprechung mit Prof. Dr. -Ing. Erwin Riederer wurde die Idee einer zusätzlichen vereinfachten Dokumentation als schnelle Nutzerhilfe zwar für erhaltungswürdig und erstrebenswert befunden, jedoch stellte sich die vorliegende my2-Version, wie der Rest der Anleitung, als ausbaufähig und unvollständig dar.

Aufgabe und damit Hauptgegenstand dieses Kapitels war es damit, die bestehende my2 Java Dokumentation zu analysieren, zu ergänzen und letztlich in die nächste Versionsstufe zu heben.

Die Ausgangslage stellte sich wie folgt dar:

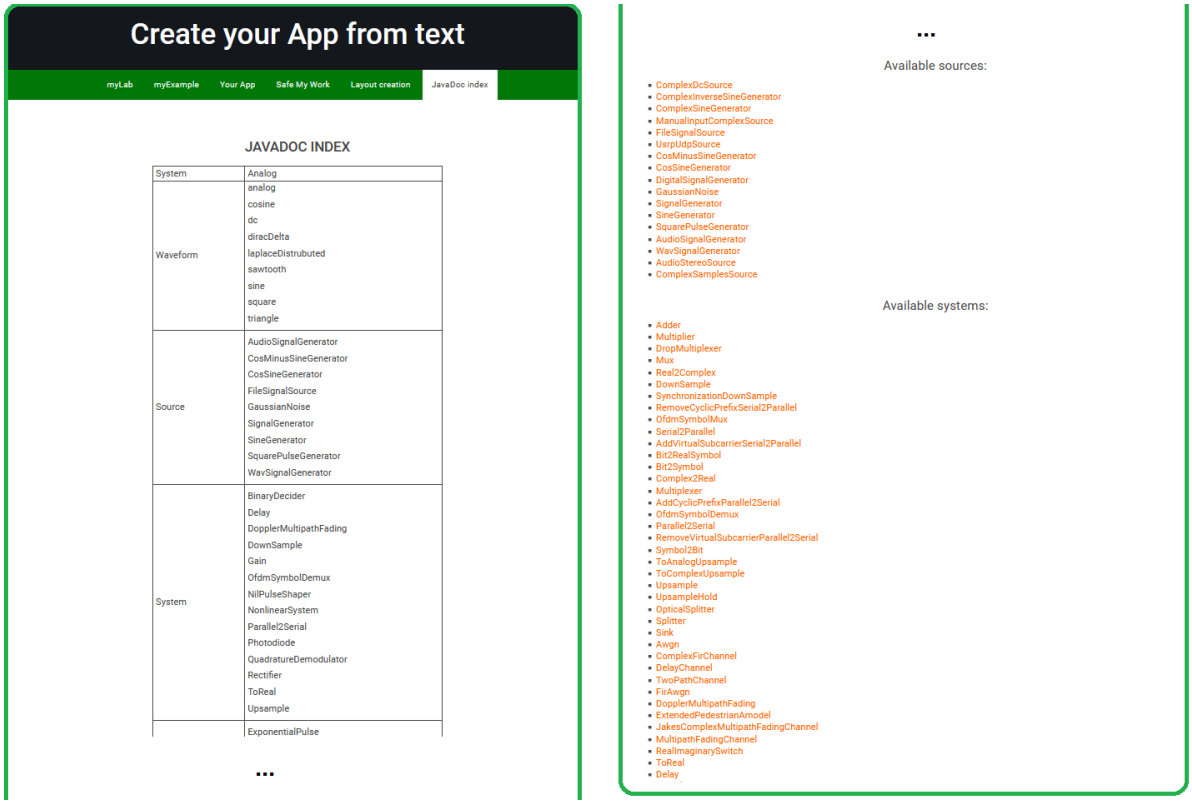


Abbildung 36 myLabAlive - JavaDocIndex -my2-Version

Abbildung 36 ist so zu verstehen, dass sich der rechte Teil der Abbildung unter der linken einordnet und nur aus Platzgründen in dieser Arbeit geteilt wurde.

In der vorherigen my2-Anleitungsversion fand sich eine Tabelle mit fünf Systemkategorien, diese waren: Waveform (=Wellenform), Source (=Quelle), System, Filter und Connector (=Verbindungsstück). In Abbildung 36 sind die ersten drei davon dargestellt. Eine jede umfasste die aus dieser Kategorie gängigsten Systemblöcke für eine eigene App-Generierung. Jedoch soll betont sein, dass es sich hierbei lediglich um die Systemblöcke aus der analogen Signaltechnik handelt.

An dieser Tabelle angehangen war eine Sammlung aller generell verfügbaren Systeme und Funktionen, im Detail 134 Aufzählungen.

Mit dieser Aufmachung waren die größten Probleme schnell identifiziert. Die größte Schwachstelle rein konzeptionell lag in der angehangenen langen Aufzählungsliste. Denn die eigentliche Idee des zusätzlichen JavaDoc Indexes war es, die bestehende maschinelle JavaDoc Dokumentation so weit zu verschmälern, dass lediglich die gängigsten und wichtigsten Systeme in einem schnell verfügbaren und ansehnlichen Abschnitt innerhalb der Anleitung zu finden sind.

Die hier bestehende Aufzählung verpasste jedoch beides dieser Ziele. Zum einen war diese lediglich alphabetisch sortierte Sammlung nicht ansehnlicher als die maschinell erzeugte von Java und zum anderen bestand kein Grund darin in zwei unterschiedlichen Verlinkungen die ein und dieselbe lange Aufzählung zu tätigen. Die Aufzählung in der my2-Version lässt sich damit schlichtweg als redundant bezeichnen.

Dass diese lange Aufzählung überhaupt in der my2-Version angehangen wurde, lässt sich nur so erklären, dass während dieser alten Überarbeitung der Wunsch aufkam, auch die gängigsten und wichtigsten Systemblöcke aus der digitalen und komplexen Signaltechnik aufzuzeigen. Diese Arbeit jedoch aufgrund des großen Zeitaufwandes hiermit unsauber abgekürzt wurde. Zumindest vermittelten Vermerkungen und Kommentare im Quellcode der Programmierung diesen Anschein. Womit auch schon die zweite Schwachstelle angesprochen wurde, die unvollständige Tabelle.

Die App-Generierung soll dem Nutzer zusätzliche Gestaltungsfreiheit innerhalb LabAlives bieten. Eine dann vermeintliche Begrenzung der Anleitung, oder zumindest des schnellen „Lexikons (JavaDoc Index)“, auf eine reine analoge Signaltechnik wäre hiermit falsch und würde dieses Ziel verfehlen.

Die bisherige JavaDoc Index Unterseite als solche musste also in Gänze überarbeitet werden.

4.6.2 Überarbeitung JavaDoc Index Auflistung

Die ersten Arbeiten widmeten sich dem Abgleich der maschinell erzeugten Java Dokumentation mit den analogen Systemen, die bereits in der Tabelle des JavaDoc Indexes enthalten waren.

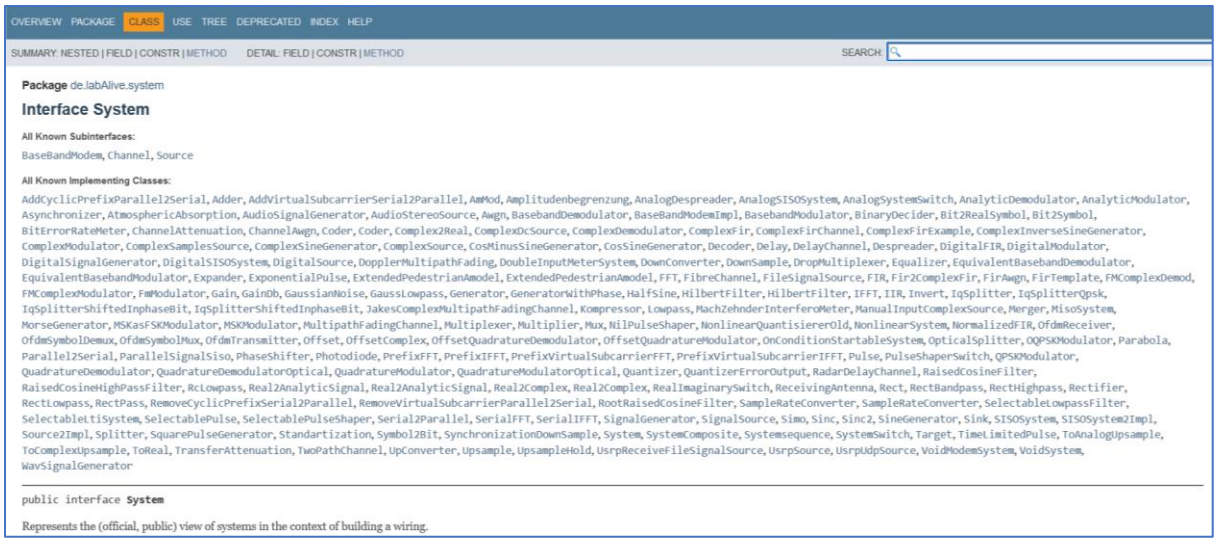


Abbildung 37 myLabAlive – maschinelles JavaDoc

Die in Abbildung 37 gezeigte maschinelle JavaDoc Version umfasste aufgrund der veralteten Aufzählung der my2-Version jedoch mehr Systeme als nur die im Anleitungsabschnitt des JavaDoc Indexes aufgelisteten. Präziser ausgedrückt waren tatsächlich 196 Systeme registriert, anstelle der im JavaDoc Index angegebenen 134.

Für den angewachsenen und aufwendigen Abgleich wurde daher erstmal die JavaDoc Aufzählung in Word überführt und tabellarisch dargestellt.

Als nächster Schritt wurden all jene Tabellenzellen entfernt, die den bereits bestehenden Systemen in JavaDoc Index entsprachen. Hierdurch blieben die reinen unsortierten und bisher für die Anleitungstabelle exklusiven Systemblöcke in Word über.

Der finale Schritt umfasste dann den eigentlichen Kern der Aufgabe. Die Analyse eines jeden übergebliebenen Systems auf Relevanz beziehungsweise Nutzbarkeit für myLabAlive und anschließender Sortierung dieses relevanten Systempools in dessen Zugehörigkeit innerhalb der Signaltechnik (analog, komplex und/oder digital).

An die bisherigen 48 analogen Systeme im JavaDoc Index wurden hierdurch weitere 38 Systeme angefügt. Diese neuen Einträge vervollständigten nicht nur den analogen Anteil in der Anleitungstabelle, sondern ergänzten sie vor allem auch um die wichtigsten komplexen und digitalen Systeme. In Summe umfasst das überarbeitete JavaDoc Index damit 86 Systeme.

Aufgrund der Masse an neu aufgelisteten Systemen musste die Tabelle jedoch neugestaltet werden. So wurde der, aus der damaligen my2-Version stammende, Systemkategoriepunkt „Waveform“ aufgelöst und dessen Inhalte an das System des SignalGenerators angehangen. Dieser ist damit ohnehin untrennbar verwand, da in diesem eben jene Waveformen selektiert und umgestellt werden.

Ebenso wurde die generelle Darstellung und Sortierung überarbeitet, um zwischen den Signaltechniken „analog, komplex und digital“ in Form von eigenen Tabellenspalten klar abzugrenzen.

Für grundlegende Systeme (beispielsweise Verbindungselemente), die in jeder Rubrik der Signaltechnik Anwendung finden, wurde hingegen auf eine Spaltenübergreifende Darstellung gesetzt. Diese neue JavaDoc Index Unterseite samt überarbeiteter Tabelle lässt sich im Anhang dieser Masterarbeit unter „Anhang 2 Neue JavaDoc Index Unterseite für myLabAlive Anleitung“ finden.

Die zuvor unter der Tabelle lang angehangene Auflistung aller Systeme wurde aufgrund Redundanz verworfen und stattdessen die allgemeinen maschinellen JavaDoc Gliederungsoberpunkte „Sources“ und „Systems“ verlinkt. Hiermit ist eine klare Trennung zwischen vollumfänglicher, maschineller JavaDoc und auf das wichtigste heruntergebrochene Lexikon des JavaDoc Index gegeben.

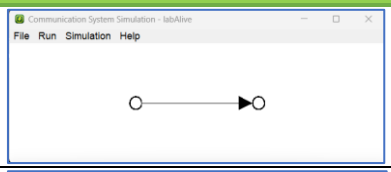
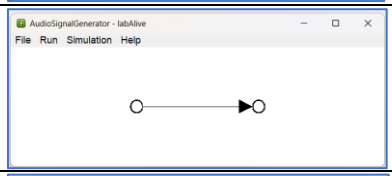
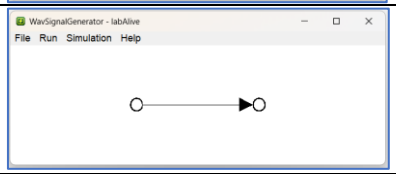
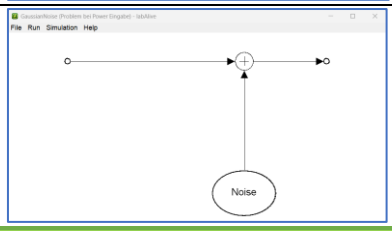
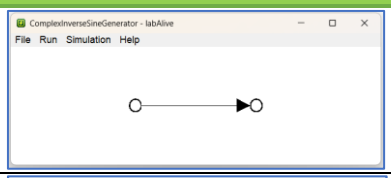
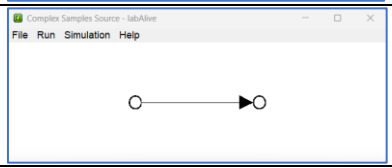
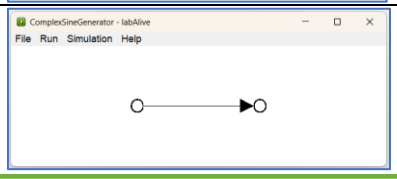
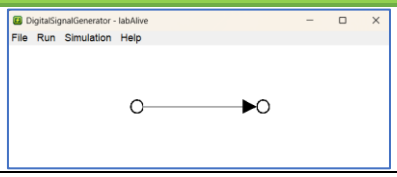
Damit sich der Nutzer trotz der angewachsenen Vielfalt auch im Umgang mit den Systemen versteht, wurden neben dieser Vervollständigung der wichtigsten Systeme in JavaDoc Index auch dessen Funktionalität in die nächste Entwicklungsstufe gehoben.

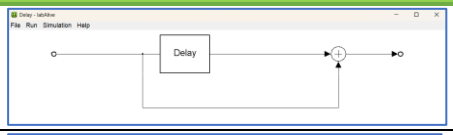
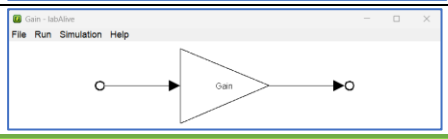
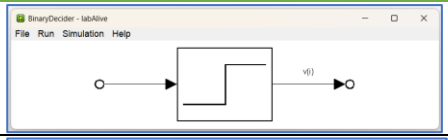
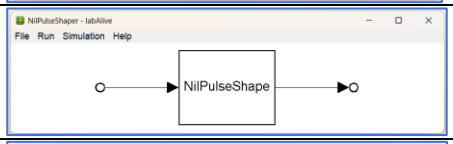
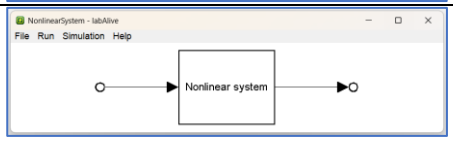
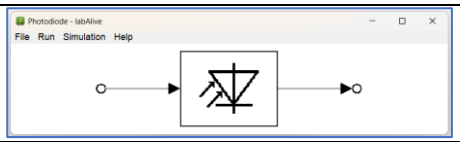
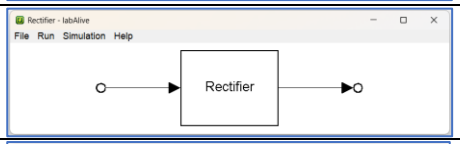
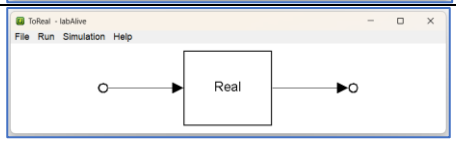
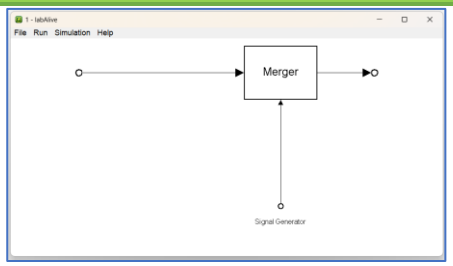
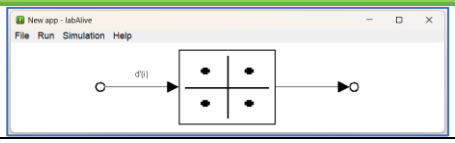
So wurde dafür gesorgt, dass die Java Doc Index Tabelle nicht nur als reine Auflistung fungiert, sondern auch für die absolut gängigsten Systeme eine Verlinkung zur Verfügung stellt, hinter der das jeweilige System samt eines kurzen Beispiels der programmiertechnischen Verwendung vorgeführt wird. Anhand der im Anhang 2 orange hervorgehobenen Systemnamen lässt sich erkennen, welche Systemnennungen damit ergänzt wurden.

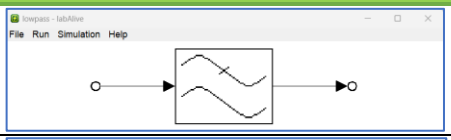
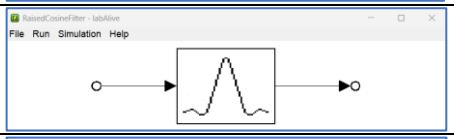
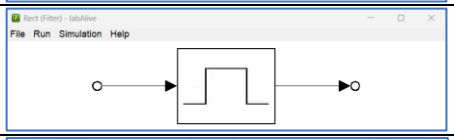
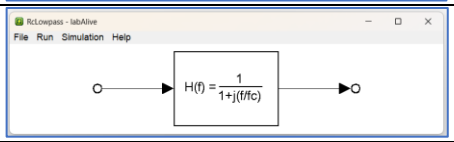
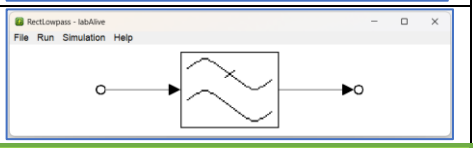
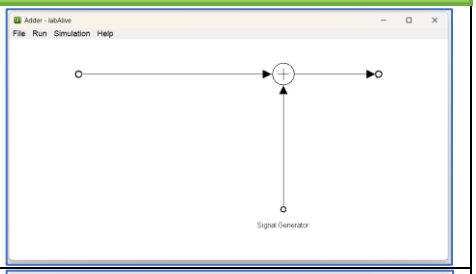
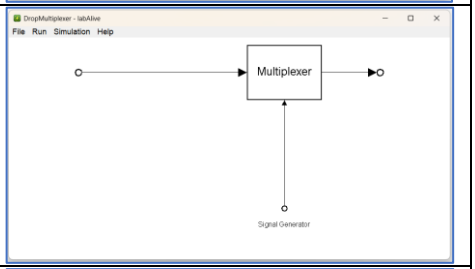
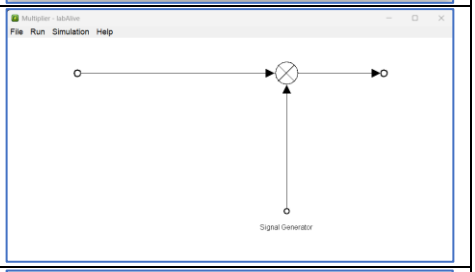
Da die Arbeiten der Bacheloranden an der neuen App-Generierung noch nicht abgeschlossen waren, wurde zur Beispielerstellung auf die alte myLabAlive Veröffentlichung gesetzt. Zugang zu dieser erfolgte über „https://www.etti.unibw.de/labalive/auth/login/“.

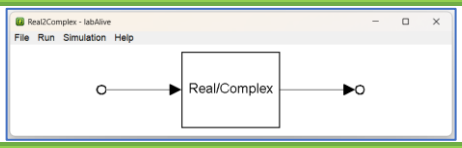
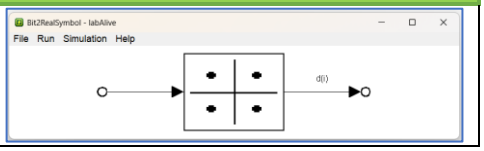
In Kapitel 4.6.3 findet sich eine Sammlung aller in JavaDoc Index eingepflegten und verlinkten kommunikationstechnischen Beispiele, die dem Nutzer als Vorzeigebeispiele dienen sollen.

4.6.3 Sammlung an ergänzten Systembeispielen in JavaDoc Index

Systemblock	***Simulationsbeschreibung***	***Blockschaltbild***
Source *(Analog)*		
SignalGenerator	signalgenerator sine(1V 5kHz) – sink	
AudioSignalGenerator	AudioSignalGenerator - sink	
WavSignalGenerator	WavSignalGenerator - sink	
GaussianNoise	sine - adder - sink gaussiannoise - adder	
Source *(Complex)*		
ComplexInverseSineGenerator	ComplexInverseSineGenerator - sink	
ComplexSamplesSource	ComplexSamplesSource(2V) - sink	
ComplexSineGenerator	ComplexSineGenerator - sink	
Source *(Digital)*		
DigitalSignalGenerator	digitalSignalGenerator - sink	

System *(übergreifend für alle Signaltechniken)*		
Delay	audiosignal - delay 0.5 - adder - sink audiosignal - adder	
Gain	signalGenerator sine - gain(2) - sink	
System *(Analog)*		
BinaryDecider	sine - BinaryDecider - sink	
NilPulseShaper	sine - NilPulseShaper - sink	
NonlinearSystem	sine - NonlinearSystem - sink	
Photodiode	sine - Photodiode - sink	
Rectifier	sine - Rectifier - sink	
ToReal (auch bei Comlex System)	ComplexInverseSineGenerator - ToReal - sink	
System *(Complex)*		
Merger	sine - Merger - sink triangle - Merger	
System *(Digital)*		
Symbol2Bit (aus complex wird digital)	ComplexSineGenerator - Symbol2Bit - sink	

Filter *(Analog)*		
lowpass	signalGenerator square - lowpass(4e3) - sink	
RaisedCosineFilter	signalGenerator sine - RaisedCosineFilter(5kHz) - sink	
Rect	signalGenerator sine - rect - sink	
RcLowpass	ComplexSineGenerator - RcLowpass - sink	
RectLowpass	sine - RectLowpass - sink	
Connector *(übergreifend für alle Signaltechniken)*		
Adder	sine - Adder - sink triangle - Adder	
DropMultiplexer	sine - DropMultiplexer - sink dc - DropMultiplexer	
Multiplier	square - multiplier - sink sine - multiplier	
Splitter	signalGenerator triangle- splitter - adder -sink splitter - lowpass(4e3) - adder	

Connector *(Analog/Complex)*		
Real2Complex	sine - Real2Complex - sink	
Connector *(Digital)*		
Bit2RealSymbol	DigitalSignalGenerator - Bit2RealSymbol - sink	

Die restlichen 57 Systeme, für die keine Anwendungsbeispiele verlinkt sind, werden wie in vorheriger Version lediglich gelistet, ohne weitere Verknüpfungen.

4.7 Überarbeitung Experiment „Analog demodulation“

Neben den Arbeiten an myLabAlive wurde sich auch mit dem Experiment zur "Analog demodulation“ aus dem Pool der LabAlive-Experimente beschäftigt.

Dieses entstand bereits während der Projektarbeit 2022 im Bachelorstudiengang und wurde damals sowohl in seiner Webpräsentation mitentwickelt als auch um ein erstmalig auf das Experiment zugeschnittenes Eingangssignal erweitert. Im Verlauf der eigenen Bachelorarbeit wurde das Experiment dann erneut erweitert und nun im Master auch als persönliches Anliegen weiterverbessert.

Sinn des Experimentes ist es, dass der Nutzer sich bei der Demodulation eines Mehrkanalsignal ausprobiert, bei dem ein jeder Unterträger ein verschieden moduliertes Signal trägt. So kann dies AM (Amplitudenmodulation), FM (Frequenzmodulation), SSB (=Einseitenbandmodulation) oder QAM (Quadraturamplitudenmoduliert) moduliert sein.

Zur Ausführung und Einleitung in das Experiment unterteilt sich dessen Webpräsentation in zwei Unterseiten, in das „Tutorial“ und in das „Experiment“.

Die Unterseite „Tutorial“ widmet sich hierbei der Versuchsvorstellung und der theoretischen Betrachtung der unterschiedlichen Modulationsarten.

Hingegen sich die Unterseite „Experiment“ der praktischen Ausführung und dem Versuchsablauf widmet.

Hauptintention der neuen Überarbeitung war es, dass der Versuch in der Zwischenzeit seit jeher um weitere zwei zu demodulierende Eingangssignale ergänzt wurde, die in die Versuchsbeschreibung mit aufgeführt und erklärt werden sollten. Und zusätzlich hierzu entstand als weitere Funktionalität für LabAlive die Option im Signalgenerator, eine sample file als neues Eingangssignal via des Servers einzuspielen. Dies galt es dem Nutzer ebenso anzuführen.

4.7.1 Einbindung der neuen Eingangssignale und allg. Seitenüberarbeitung

Das Experiment "Analoge Demodulation" enthält eine App, bei der der Anwender gemäß den damaligen Standards das zu demodulierende Signal selbst einfügen musste. Dies geschah durch vorherigen Signaldownload von der Experimentenseite.

Durch die zwischenzeitliche Weiterentwicklung von anderer Seite entsprach dies jedoch nicht mehr dem aktuellen Stand. Denn mit der neuen Version wurde das ohnehin fest darauf zugeschnittene und zu verwendende Signal nun fest in den Signalgenerator-Funktionsblock der App eingefügt, um dem Nutzer den beschriebenen manuellen Einfügeschritt zu ersparen.

Des Weiteren wuchs das Signalportfolio, wie bereits in Kapitel „4.7 Überarbeitung Experiment „Analog demodulation““ beschrieben, in der Zwischenzeit um zwei weitere Signale an, welche zwar nun ebenso bereits fest im Funktionsblock des Signalgenerators eingebunden waren, jedoch noch nicht in der Experimenten-Unterseite auf LabAlive erwähnt wurden. Und auch die Aufmachung der App veränderte sich, so wurde der Adder-Baustein der Schaltung in der Zwischenzeit grafisch überarbeitet.

Zur Besserung wurde unteranderem der Seitenanfang der Experimenten-Unterseite auf die nun im Signalgenerator eingebundene Signale ausgerichtet und der Nutzer über die seit Neuem drei bestehenden Signale und über die Möglichkeit der Signaleinfügung via Server unterrichtet. Einen Vorher/Nachher Vergleich samt grafisch überarbeitetem Adder-Schaltungsbaustein findet sich in Abbildung 38. Es sei betont, dass auch alle weiteren Grafiken der Unterseite für diesen neuen Schaltungsbaustein überarbeitet wurden, exemplarisch soll im Folgenden jedoch der gezeigte Ausschnitt mit Abbildung 38 genügen.

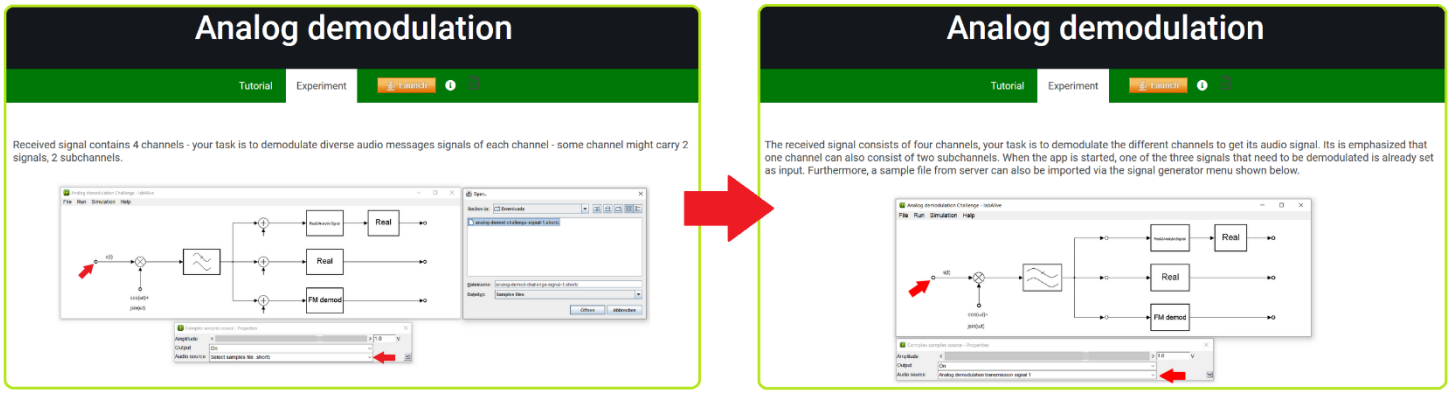


Abbildung 38 Vorher/Nachher Vergleich der Experimenten-Unterseite der "Analog demodulation"

Des Weiteren deckte bisher, wie in nachfolgender Abbildung gezeigt, die Demodulations-Musterlösung lediglich das erste Signal ab, während der Nutzer keine Selbstkontrolle über den Demodulationsweg der Signale zwei und drei durchführen konnte.

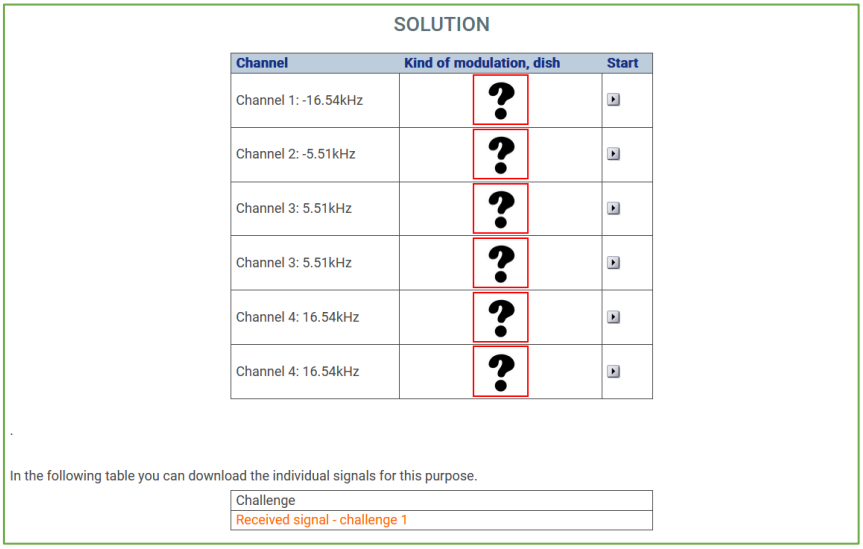


Abbildung 39 Ausschnitt der ehemaligen Experimenten-Unterseite der "Analog demodulation"

Zugleich existierte am Ende der Seite auch noch der mittlerweile obsolete Link für den Signaldownload des ersten Signals für den beschriebenen manuellen Einfügeschritt.

Letzteres wurde, aufgrund des vereinfachten Nutzungsablauf der App, ersatzlos entfernt und zugleich Raum für die anzuwachsende Musterlösung geschaffen.

Aus Gründen der Übersichtlichkeit wurde sich entschieden, die bereits bestehende Musterlösungstabelle des ersten Signals nicht zu erweitern, sondern für die weitere Selbstkontrolle eine weitere Tabelle im identischen Stil anzufügen.

Das erste aufzunehmende Signal entstand zu Zeiten der damaligen Bachelorarbeit und stellte sich spektral und in seiner Zusammensetzung wie folgt dar:

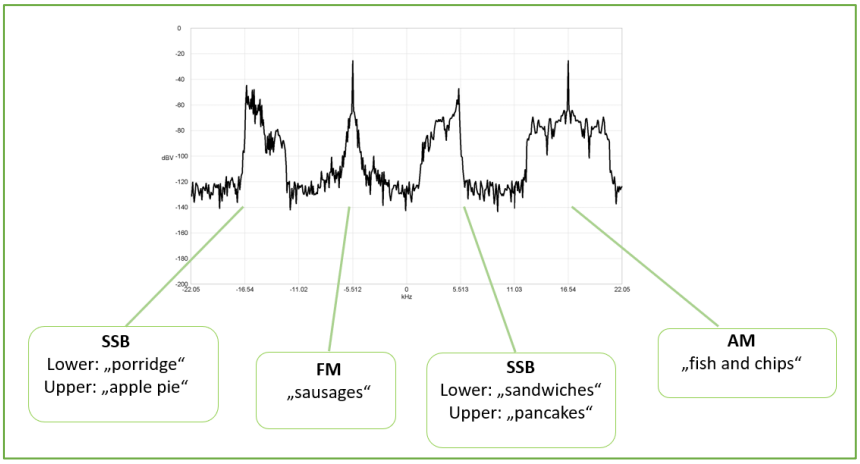


Abbildung 40 Spektrumsansicht "Analog demodulation transmission signal 2" der damaligen Bachelorarbeit

Da die enthaltenen Lösungswörter, deren Kanalposition und die jeweilige Modulationsart klar dokumentiert waren, konnte hierauf zügig aufgebaut werden und die Lösungstabelle, angelehnt an Abbildung 39 zügig generiert werden.

Das letzte aufzunehmende Signal 3 ließ sich zwar trotz fehlender Dokumentation richtig demodulieren, jedoch warf die Signalzusammensetzung Fragen auf. So setzte sich dieses zwar ebenso aus vier Kanälen mit den identischen Trägerfrequenzen zusammen, jedoch wurde von einem Modulations- und Audiowechsel abgesehen. Jedes der vier Kanäle weiste die identische Audio-Musikspur auf und wurde in drei Fällen als oberes single-sideband (USB) und in einem Fall als unteres single-sideband (LSB) moduliert. Hier fehlte also die eigentlich vom Experiment angestrebte herausfordernde Signalvielfalt. Dies wird in der Spektrumsansicht in Abbildung 41 ersichtlich. In diesem zeigen sich die vier Kanäle und ihr zueinander identischer Inhalt.

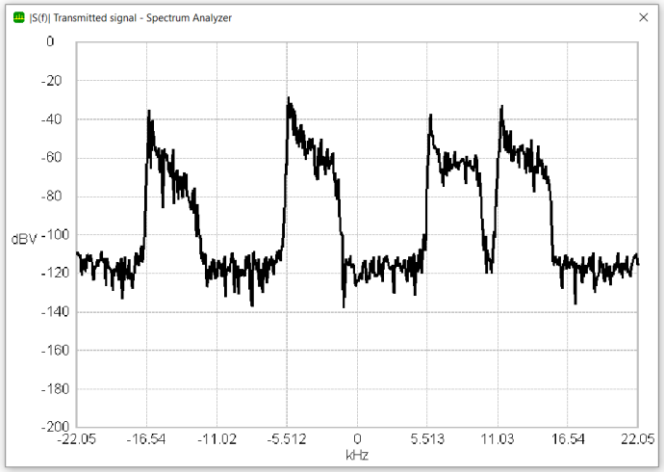


Abbildung 41 Spektrum des " Analog demodulation transmission signal 3"

Nach eigener erfolgter Bewertung wurde dieses Signal damit weiterhin aus der Musterlösung außen vorgelassen, da es sich hierbei wohl um eine fehlerhafte Einbindung von anderer Seite handeln musste. Zur Menge der zu demodulierenden Signale blieb es jedoch dennoch gezählt, damit dieser Fehler auf absehbare Zeit von verantwortlicher Seite nachgebessert werden kann, ohne den Inhalt der Experiment-Unterseiten über die Musterlösung hinaus anpassen zu müssen.

5. Fazit

Zusammenfassend lässt sich anführen, dass die unter Kapitel 1.3 beschriebene Aufgabenstellung in Gänze erfolgreich bewerkstelligt wurde. So konnte nicht nur die Dokumentation und Einführung des „myLabAlive“s aufgebessert und mit sinnvollen kommunikationstechnischen Showcase Beispielen aufgewertet werden, sondern auch noch das in Kapitel 4.6 beschriebene JavaDoc Index um die komplexe und digitale Ebene erweitert und mithilfe einer neuen Funktionalität in die nächste Entwicklungsstufe gehoben werden. Der in seiner Arbeitszeit umfassendste Unterpunkt bildete jedoch die Planung und Verwirklichung des unter Kapitel 4.4 beschriebenen Tutorialvideos. So stellt dies in Zukunft nicht nur die Nutzerhilfe für den Umgang mit dem „myLabAlive“-Bereich und der Vorstellung der interaktiven Appinternen Speicherfunktion dar, sondern wurde obendrein auch noch zum Prunkstück dieser neuen Anleitung.

Neben diesen Arbeiten konnte parallel auch noch das bereits bestehende Experiment „Analog demodulation“ aus dem LabAlive-Experimentenpool, sowohl in seiner Gestaltung als auch in seinem Inhalt ausgebaut und auf den aktuellen Stand gebracht werden (siehe Kapitel „4.7 Überarbeitung Experiment „Analog demodulation““).

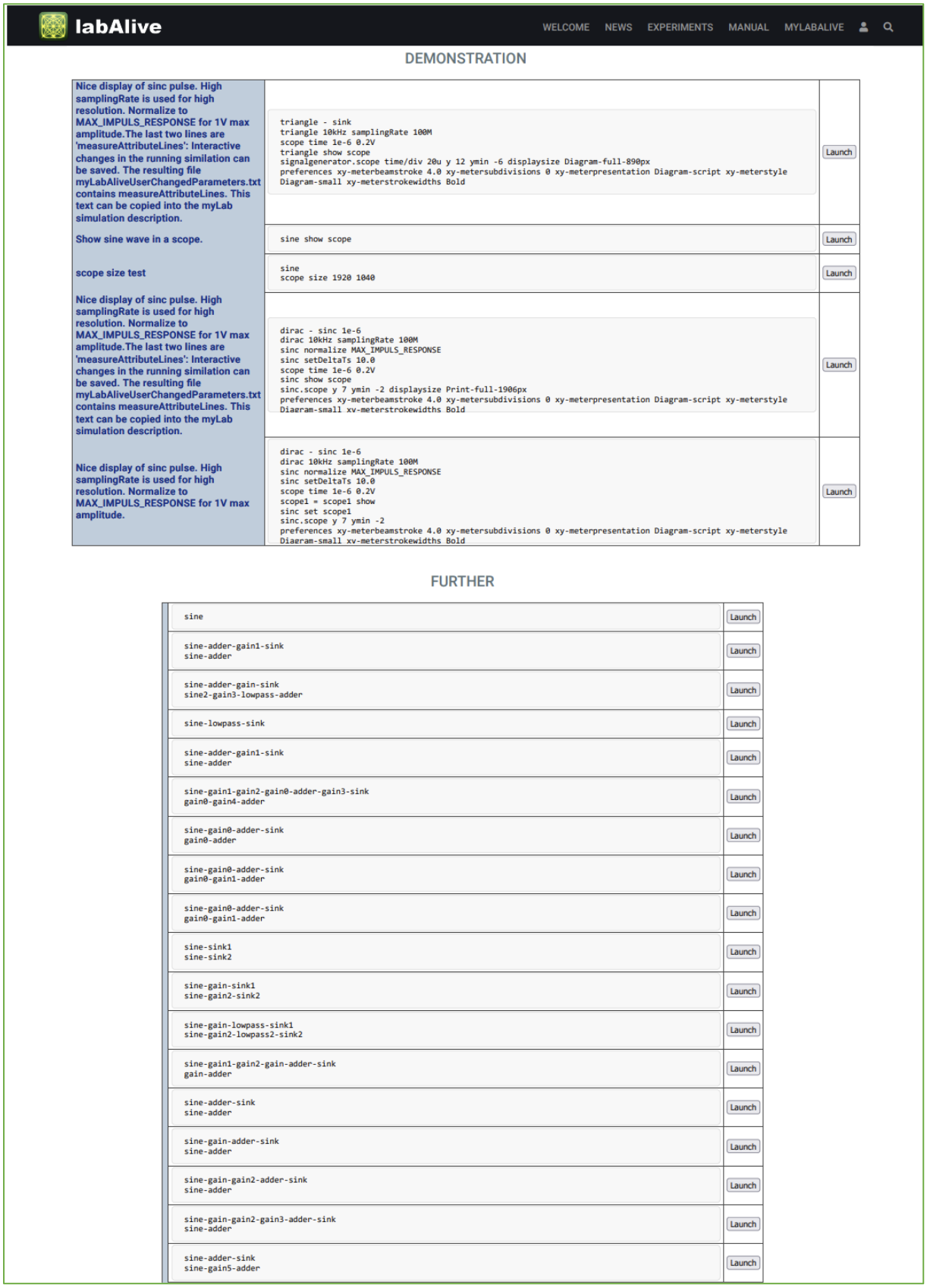
6. Ausblick

Wie bei dieser Arbeit bereits das Nacharbeiten im Experiment der „Analog demodulation“ aus dem LabAlive-Experimentenpool von Nöten war, so werden auch in Zukunft die Bereiche LabAlive´s aktualisiert werden müssen. Schon heute lässt sich abzeichnen, dass vor allem der behandelte „myLabAlive“-Bereich Gegenstand einer raschen Weiterentwicklung sein wird und damit auch die angefertigte Anleitung einer Anpassung erfordert. Aus diesem Grund beschränkt sich das aus dieser Masterarbeit hervorgehende Tutorialvideo auch auf die grundlegenden Funktionen der überarbeiteten Nutzerumgebung, um einer möglichst langen Aktualität gerecht zu werden.

Zudem kann der in Kapitel „4.7.1 Einbindung der neuen Eingangssignale und allg. Seitenüberarbeitung“ beschriebene Signalfehler innerhalb der „Analog demodulation“ Simulation von verantwortlicher Seite nach deren ursprünglichen Vorstellungen zügig behoben werden. Hierdurch kann dann auch die Musterlösung auf der betroffenen Unterseite final zu einem Abschluss gebracht werden.

Dies schließt damit den Tätigkeitsbereich dieser Masterarbeit ab.

7. Anhang



Anhang 1 Ausschnitt aus Example-Sammlung der Urdokumentation von MyLabAlive

Create your App from text

myLab | myExample | Layout creation | Your App | JavaDoc index

JAVADOC INDEX

Block	Analog	Complex	Digital
Source	SignalGenerator *sine* (SineGenerator) *cosine* *triangle* *sawtooth* *square* (SquarePulseGenerator) *dc* *randomSquare* *laplaceDistrubuted* *diracDelta* AudioSignalGenerator WavSignalGenerator CosMinusSineGenerator CosSineGenerator GaussianNoise FileSignalSource	AudioStereoSignalGenerator ComplexSource ComplexInverseSineGenerator ComplexSamplesSource ComplexSineGenerator ManualInputComplexSource	DigitalSignalGenerator DigitalSource
System	Delay DopplerMultipathFading Gain Upsample		
	BinaryDecider DownSample FmModulator JakesComplexMultipathFadingChannel OfdmSymbolDemux NilPulseShaper NonlinearSystem Parallel2Serial Photodiode QPSKModulator QuadratureDemodulator Rectifier SampleRateConverter Serial2Parallel ToReal	AnalyticDemodulator AnalyticModulator ComplexDemodulator ComplexModulator EquivalentBasebandDemodulator EquivalentBasebandModulator FMComplexDemod FMComplexModulator IqSplitter IqSplitterShiftedInphaseBit IqSplitterQpsk Merger QuadratureModulator SampleRateConverter ToReal	AddVirtualSubcarrierSerial2Parallel BinaryDecider BitErrorRateMeter DownSample QuadratureDemodulatorOptical QuadratureModulatorOptical SampleRateConverter Symbol2Bit
Filter	ExponentialPulse GaussLowpass IRR Lowpass RaisedCosineFilter RcLowpass Rect RectBandpass RectHighpass RectLowpass TimeLimitedPulse	Fir2ComplexFir IIR	DigitalFIR
Connector	Adder DropMultiplexer Multiplexer Multiplier Sink Splitter		
	Complex2Real Mux Real2Complex	Complex2Real Real2Complex	Bit2RealSymbol

Further available blocks

All available sources
All available systems

Anhang 2 Neue JavaDoc Index Unterseite für myLabAlive Anleitung

8. Literaturverzeichnis

[1] “Average Screen Time in US vs. the rest of the world” von Rebecca Moody
Bericht zu Digital 2023: Global Overview Report
Online verfügbar unter:
https://www.comparitech.com/tv-streaming/screen-time-statistics/
abgerufen am 04.07.2023

[2] “What Is Eclipse?”
Produktbeschreibung der Eclipse Foundation
Online verfügbar unter:
https://www.eclipse.org/home/whatis/
abgerufen am 05.07.2023

[3] “Eclipse EGit: Git Integration for Eclipse”
Produktbeschreibung der Eclipse EGit
Online verfügbar unter:
https://projects.eclipse.org/projects/technology.egit
abgerufen am 05.07.2023

[4] “What is OBS Studio? [Complete Guide]”
Produktbeschreibung von OBS Studio durch movavi
Online verfügbar unter:
https://www.movavi.com/learning-portal/what-is-obs-studio.html
abgerufen am 16.07.2023

[5] “DaVinci Resolve”
Produktbeschreibung von DaVinci Resolve durch PCMagazine UK
Online verfügbar unter:
https://uk.pcmag.com/video-editing/134729/davinci-resolve
abgerufen am 02.08.2023

[6] Skript „Telekommunikationstechnik“ von Prof. Dr. -Ing. Erwin Riederer
Universität der Bundeswehr München
Fakultät Elektrotechnik und Technische Informatik (ETTI)
Kapitel 1.4 Nachrichtenübertragung, Seite 3-9
Skriptversion Stand 07.01.2016

[7] „Fundamentals of Communication Systems” von John G. Proakis & Masoud Salehi
Pearson Verlag, Second global Edition 2014
ISBN-13: 978-1-292-01568-2
Kapitel 3.2 „Amplitude Modulation“, Seite 139ff.

[8] Skript „Telekommunikationstechnik“ von Prof. Dr. -Ing. Erwin Riederer
Universität der Bundeswehr München
Fakultät Elektrotechnik und Technische Informatik (ETTI)
Kapitel 4 Amplitudenmodulation/-demodulation, Seite 23-30
Skriptversion Stand 07.01.2016

[9] Skript „Telekommunikationstechnik“ von Prof. Dr. -Ing. Erwin Riederer
Universität der Bundeswehr München
Fakultät Elektrotechnik und Technische Informatik (ETTI)
Kapitel 7 Frequenzmodulation/-demodulation, Seite 38-44
Skriptversion Stand 07.01.2016

[10] „FM-Demodulation”
DARC-Online-Lehrgang Technik Klasse A Kapitel 12: Modulation und Demodulation
Online verfügbar unter:
https://www.darc.de/der-club/referate/ajw/lehrgang-ta/a12/#FM_Demodulation
abgerufen am 02.08.2023

[11] „Einseitenbandmodulation”
Lerntutorial ESB-AM von der „Lehr– und Forschungseinheit für Nachrichtentechnik der Technischen Universität München“
Online verfügbar unter:
https://www.lntwww.de/Modulationsverfahren/Einseitenbandmodulation
abgerufen am 02.08.2023

[12] “Hilbert Transform”
Hilbert Transform Documentation in MathWorks
Online verfügbar unter:
https://de.mathworks.com/help/signal/ug/hilbert-transform.html
abgerufen am 02.08.2023

[13] „Synchrondemodulation”
Lerntutorial Synchrondemodulator von der „Lehr– und Forschungseinheit für Nachrichtentechnik der Technischen Universität München“
Online verfügbar unter:
https://www.lntwww.de/Modulationsverfahren/Synchrondemodulation
abgerufen am 03.08.2023

[14] „Quadratur–Amplitudenmodulation (QAM)”
Lerntutorial Quadratur–Amplitudenmodulation (QAM) von der „Lehr– und Forschungseinheit für Nachrichtentechnik der Technischen Universität München“
Online verfügbar unter:
https://www.lntwww.de/Modulationsverfahren/Weitere_AM%E2%80%93Varianten#Quadratur.E2.80.93Amplitudenmodulation_.28QAM.29
abgerufen am 03.08.2023

[15] „Weißes gaußsches Rauschen”
Physiklexikon von Hans-Peter Willig mit Verweis auf die Enzyklopädie Wikipedias
Online verfügbar unter:
https://www.cosmos-indirekt.de/Physik-Schule/Additives_wei%C3%9Fes_gau%C3%9Fsches_Rauschen
abgerufen am 07.08.2023