DHBW
Duale Hochschule
Baden-Württemberg
Mannheim

THEMA STUDIENARBEIT

Erweiterung eines virtuellen Oszilloskops

im Studiengang

Elektrotechnik

an der *Dualen Hochschule Baden-Württemberg Mannheim*

von

Name, Vorname	Epperlein, Justus
Abgabedatum	02.04.2023
Bearbeitungszeitraum	09.01.-02.04.2023
Matrikelnummer, Kurs	1527321, TEL20AT1
Ausbildungsfirma	ABB AG
Betreuer	Prof. Dr.- Ing. Rüdiger Heintz

Erklärung zur Eigenleistung

Ich versichere hiermit, dass ich meine Projektarbeit mit dem Thema: Erweiterung eines virtuellen Oszilloskops selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

_____ _____

Ort, Datum Unterschrift

Abstract

Der Fortschritt von Simulatoren in der Nachbildung von technischen Prozessen erlaubt heutzutage die realitätsnahe Durchführung von Laborversuchen in digitalen Anwendungen. Diese Möglichkeit der Digitalisierung von Laborumgebungen ist für Hochschulen aus Gründen der Verfügbarkeit und der Kosten sehr attraktiv. Diese Studienarbeit beschreibt die Überarbeitung und die Erweiterung einer Frontend-Webanwendung, in der Messwerte aus dem eingebundenen Schaltungssimulator CircuitJS auf einem virtuellen Oszilloskop abgebildet werden. Die Anwendung soll eines Tages im Grundlagenlabor Elektrotechnik zur Vermittlung des Umgangs mit Oszilloskopen eingesetzt werden.

Die Bearbeitung der Anwendung umfasste unter anderem die Modifizierung der Erstellung der Bildelemente, aus denen sich das Modell des virtuellen Oszilloskop zusammensetzt. Um die Anordnung der grafischen Komponenten auf unterschiedlichen, sich überschneidenden Ebenen zu realisieren, wurden sie auf mehreren Canvas-Elementen verteilt. Zudem konnte der Datenübergang von der Simulation in die Anwendung, sowie die Ablage und Nutzung der Messwerte in einem Ringspeicher konzipiert und umgesetzt werden. Dieser Speicher dient wiederum als Grundlage für die überarbeitete reguläre Zeichenfunktion und im Triggermodus für das Zeichnen der getriggerten Signale, nachdem eine Auswertung der Daten auf Triggerereignisse in einem dafür angelegten Web Worker erfolgt ist. Für den Trigger wurden die Modi Edge und Edge then Edge mit den zugehörigen Konfigurationsoptionen implementiert. Außerdem konnte die Programmierung des Oszilloskopmenüs und dessen Darstellung umgestaltet werden. Zusätzlich wurden die lokalen Dateien der CircuitJS-Applikation aktualisiert. Durch das Projekt wurden die vorgegebenen Erweiterungen des virtuellen Oszilloskops erfolgreich durchgeführt und eine gute Basis für künftige Weiterentwicklungen geschaffen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Quellcodeverzeichnis

1 Projektbeschreibung

Die Studienarbeit beschreibt einen Abschnitt der Entwicklung eines virtuellen Oszilloskops als Frontend-Webanwendung. Diese Webanwendung wurde bereits in einer vorangegangenen Studienarbeit erstellt und beinhaltet den Schaltungssimulator CircuitJS, der das individuelle Zusammenstellen von elektrischen Schaltungen, die Visualisierung und Simulation dieser erlaubt. Es ist vorgesehen, dass gewisse Messdaten aus der CircuitJS Simulation in das Oszilloskop übertragen werden, wo sie darzustellen und auszuwerten sind. Das endgültige Ziel dieses Entwicklungsprozesses ist die Erstellung einer Anwendung, die von Studenten im Grundlagenlabor Elektrotechnik verwendet werden kann. Dies würde den Vorteil haben, dass keine realen, kostenintensiven Oszilloskope erforderlich wären. Außerdem bestünde für die Geräte nicht die Gefahr der Beschädigung durch inkorrekte Bedienung.

Bestimmte Funktionen eines Oszilloskops sind im Zuge dieser Studienarbeit umzusetzen. Dazu gehören hauptsächlich die Datenaufnahme und die fundamentale Datenverarbeitung für die gewünschten Signale aus der Simulation. Konkret wird sowohl die reguläre Visualisierung der Signale, sowie die Abbildung der Signale im Triggermodus bearbeitet. Um diese Funktionen konfigurierbar zu machen, wird angestrebt, dass im Optionsmenü des virtuellen Oszilloskops ausgewählte Einstellungen bestimmt werden können. Die Speicherung der getroffenen Einstellung ist zu implementieren und die Darstellungsfunktionen auf eine Weise zu programmieren, dass sie sich den Einstellungen anpassen. Das Fundament für den Betrieb des virtuellen Oszilloskops bilden die Bildelemente, welche in Abschnitt 2.4.1 beschrieben werden. Die Überprüfung der Deklaration dieser Objekte muss ebenfalls mit diesem Projekt durchgeführt werden. Ein Teil der genannten Funktionen wurde bereits in der Vergangenheit bearbeitet, konnte aber nicht abgeschlossen werden und ist weitestgehend fehlerbehaftet oder eine anderweitige Umsetzung wird gefordert.

2 Vorbetrachtungen

Bevor auf die Durchführung eingegangen wird, werden im Folgenden einige Thematiken beleuchtet, die für die Projektbearbeitung relevant sind. Dazu werden zuerst die Grundlagen der Webentwicklung erläutert und danach das Oszilloskop als Werkzeug in der Elektrotechnik vorgestellt. Das anschließende Kapitel 2.3 beschäftigt sich mit dem Schaltungssimulator CircuitJS, der in der Anwendung eingebunden ist. Außerdem wird der Stand der Anwendung geschildert, der für die Arbeit als Grundlage dient. Zuletzt wird die Vorgehensweise für die Durchführung des Projekts beschrieben.

2.1 Grundlagen der Webprogrammierung

Die Frontend-Webentwicklung beschreibt die Erstellung und Bearbeitung eines Webservers, der im Browser des Bedieners ausgeführt wird und diesem eine Benutzeroberfläche erzeugt. Frontend-Websites sind aus Hypertext Markup Language (HTML), Cascading Style Sheets (CSS) und JavaScript Dateien zusammengesetzt [Vgl. 1]. Grundsätzlich ließen sich Webseiten auch vollständig mit HTML erstellen, CSS und JavaScript erweitern aber deren Gestaltungsmöglichkeiten und Funktionalitäten enorm [Vgl. 2, S. 31 ff.], weshalb alle dieser Sprachen im Folgenden näher erläutert werden.

2.1.1 HTML

Der im Begriff HTML enthaltene Hypertext beschreibt einen Text, der Querverweise besitzt [Vgl. 3, S. 57] und als Auszeichnungssprache verwendet HTML keine Befehle [Vgl. 4, S. 26], sondern „besteht aus Elementen, die wiederum aus Start- und End-Tags sowie aus Inhalt bestehen.“ [3, S. 89] „HTML dient [...] dazu, den Aufbau einer Webseite zu strukturieren“ [3, S. 65], indem Dokumente gestaltet und mit Überschriften, Text, Tabellen, Listen etc. ausgestattet werden [Vgl. 4, S. 26]. „HTML-Dokumente können von jedem Webbrowser dargestellt werden und gelten somit auch als Basis für das Internet.“ [5, S. 46] Wird ein HTML-Dokument in Kombination mit CSS und JavaScript Dateien eingesetzt, dann muss im Dokument auf diese Dateien verwiesen werden [Vgl. 3, S. 71]. Die HTML-Dokumente lassen sich dabei mit verschiedensten Texteditoren bearbeiten [Vgl. 3, S. 58].

2.1.2 CSS

Während die Strukturierung einer Webseite mit HTML erfolgt, wird die Formatierung bzw. Gestaltung mit CSS durchgeführt [Vgl. 2, S. 30]. Auf diese Weise wird eine Trennung von Inhalt und Optik vorgenommen [Vgl. 2, S. 32]. Diese Differenzierung hat den Vorteil, dass die Navigation in den Anwendungsdateien, sowie spätere Änderungen erleichtert werden und im Vergleich zu HTML in CSS mehr Gestaltungsmöglichkeiten zur Verfügung stehen [Vgl. 6]. Wenn externe CSS Dateien dem Browser Gestaltungsanweisungen für die Webseite vorgeben [Vgl. 3, S. 66], führt dies zu übersichtlicheren Quellcodes und die CSS-Dateien lassen sich auch für mehrere HTML-Dokumente gleichzeitig einbinden [Vgl. 2, S. 30].

CSS gestaltet die Website über eine Sammlung von Formatvorlagen [Vgl. 4, S. 27]. Zur Bestimmung der Darstellungseigenschaften eines Elements werden in CSS Regeln angelegt, die aus einem Selektor und einem Deklarationsblock bestehen. Die Deklaration umfasst eine Eigenschaft, wie z.B. color, und einen Wert, die durch einen Doppelpunkt voneinander getrennt sind [Vgl. 4, S. 27]. „Selektoren legen [...] fest, auf welche Elemente im Dokument eine Formatierung angewendet werden soll.“ [4, S. 27] Darüber hinaus bietet CSS weitere Funktionen, um die funktionale Trennung zwischen Inhalt und Gestaltung zu gewährleisten [Vgl. 3, S. 71].

Für die Eigenschaften eines HTML-Elements gibt es browserabhängige Standardwerte, die sich mit CSS anpassen lassen [Vgl. 4, S. 27]. Die Darstellung eines HTML-Dokuments kann deshalb zwischen unterschiedlichen Browsern variieren [Vgl. 4, S. 35], aber mit CSS lässt sich eine einheitliche Darstellung erreichen, wobei es eher das Ziel der Webprogrammierung ist, „den Inhalt an jedes System korrekt auszuliefern und die zugehörige visuelle Präsentation entsprechend den Fähigkeiten des verwendeten Browsers zu ermöglichen.“ [4, S. 35]

2.1.3 JavaScript

„JavaScript ist eine Programmiersprache, die speziell für das Web entwickelt wurde [...] und dient vor allem dazu, Eingaben des Nutzers zu verarbeiten und Elemente auf Webseiten zu verändern.“ [3, S. 243] Gegenüber den restlichen Komponenten der Frontend-Webprogrammierung ist JavaScript demnach für die Interaktivität und das Verhalten der Webseite verantwortlich. Mithilfe von JavaScript „lassen sich ’Verhaltensänderungen’ an HTML-Elementen erreichen.“ [2, S. 33] und mit JavaScript „übertragen

Webseiten Daten, ohne gleich die ganze Seite neu zu laden (man spricht von *asynchroner Datenübertragung* [...])." [3, S. 243] Dies wird für Animationen, zur Verbesserung der User-Experience, zur Reaktion auf Anwenderoptionen und zur Veränderung des Aussehens der Webseite eingesetzt, weshalb JavaScript bei fast allen Websites Verwendung findet [Vgl. 2, S. 33]

2.2 Oszilloskop

Ein Oszilloskop ist ein elektronisches Gerät und Werkzeug in der Elektrotechnik zur Untersuchung von Spannungsverläufen, indem diese auf einem Bildschirm als Spannungs-Zeit-Diagramme visualisiert werden und das Oszilloskop verschiedenen Optionen bietet, die Signale auszuwerten. Ein Oszilloskop verfügt über zwei bis vier Eingangskanäle, um Spannungen an verschiedenen Stellen einer Schaltung über Tastköpfe aufzunehmen und zu vergleichen. Die Tastköpfe verfügen dabei teilweise über eigene Masseanschlüsse, sodass sich nicht nur elektrische Potenziale, sondern beispielsweise auch die Spannungsabfälle über gewissen elektrischen Komponenten messen lassen. Die Messwerte der verschiedenen Eingänge werden auf der Anzeige des Oszilloskops durch unterschiedliche Farben differenziert. Bedeutend für dieses Projekt ist die Triggerfunktion eines Oszilloskops, bei der die Signale nur gezeichnet werden, wenn sie ein definierbares Kriterium erfüllen, wie z.B. einen Grenzwert überschreiten. Das Signal wird jeweils bei der Erfüllung dieses Kriteriums dargestellt und der Zeitpunkt des Auslösens befindet sich dabei in einer vorgebbaren Position auf der x-Achse.

Als Vorlagen für das virtuelle Oszilloskop dienen die Oszilloskope der 3000 X-Serie von Keysight Technologies, bei denen es sich um professionelle Vier-Kanal-Oszilloskope handelt [Vgl. 7]. Die Kosten für ein solches Oszilloskop können 6000 Euro übersteigen [Vgl. 7], weshalb es erstrebenswert ist, die Einarbeitung der Studierenden im Grundlagenlabor Elektrotechnik an virtuellen Oszilloskopen vorzunehmen. Damit dieser Lernprozess effektiv abläuft, ist es aber auch sinnvoll, das virtuelle Oszilloskop sowohl optisch als auch funktional dem realen Gerät möglichst genau nachzubilden.

2.3 CircuitJS

CircuitJS ist ein auf JavaScript basierender Schaltungssimulator, der ursprünglich von Paul Falstad geschaffen wurde und die unkomplizierte Erstellung und Simulation di-

verser elektrischer Komponenten und Schaltungen ermöglicht [Vgl. 8, S. 1]. Die Applikation kann in Abbildung 3 betrachtet werden und verfügt über einige Möglichkeiten, Schaltungen zu importieren oder zu exportieren. Daneben bietet sie verschiedene Bearbeitungs- und Anzeigeoptionen. Über den Reiter Zeichnen lassen sich eine Vielzahl elektrischer Bauteile einfügen, darunter Wiederstände, Induktivitäten, Kapazitäten, Transistoren, Energiequellen und Schalter, aber auch gewisse integrierte Schaltungen. Diese Bauteile können über Leitungen frei verbunden und kombiniert werden. Die Anwendung kann selbst Spannungswerte als Graphen in Oszilloskop-Oberflächen darstellen, diese lassen sich aber nur sehr eingeschränkt konfigurieren und bieten keine weiteren Funktionen eines Oszilloskops. Der Simulator verfügt über einige Einstellungen, hauptsächlich der Schaltungsanzeige betreffend. Zudem lassen sich mit dem Reiter Schaltungen Elemente einer umfangreichen Auswahl von vorgefertigten Standardschaltungen öffnen. Im Simulationssteuerungsbereich ist es möglich, die Simulation zurückzusetzen oder zu unterbrechen. Außerdem können dort die Simulationsgeschwindigkeit und die Stromgeschwindigkeit über Schieberegler verändert werden.

Im Bereich der professionellen Schaltungssimulatoren gibt es eine Vielzahl von vergleichbaren Applikationen. Die bekanntesten davon sind LTspice von Analog Devices und Multisim von National Instruments, welche beide auf der Simulationssoftware SPICE für elektrische Schaltungen aufbauen [Vgl. 9, S. 28]. Beide bieten eine größere Auswahl an elektrischen und elektronischen Komponenten zur Schaltungserstellung, sowie an Bedienoptionen [Vgl. 9, S. 27]. In Multisim ist sogar die Simulation der beiden Oszilloskope Modelle Agilent 54622D und Tektronix TDS 2024 bereits verfügbar [Vgl. 9, S. 120 ff.]. Für das Grundlagenlabor Elektrotechnik wäre allerdings eine einfach zu bedienende Software interessanter. Da sich CircuitJS relativ leicht in die Webanwendung des virtuellen Oszilloskops einbinden lässt, ist der Simulator besonders geeignet für das Projekt.

2.4 Aktueller Stand der Anwendung

2.4.1 Anwendungsoberfläche

Die Anwendungsoberfläche besteht aus drei Bereichen, dem virtuellen Oszilloskop an sich, der Testsektion mit weiteren Bedienelementen und Eingabefeldern [Vgl. 10, S. 16 f.] und dem Simulationsfenster. Das angedeutete Modell eines Oszilloskops, das in Abbildung 1 dargestellt ist, besteht aus einem Bildschirm und mehreren Tasten. Ein

Teil des Bildschirms wird von der Signalanzeige eingenommen, in deren Bereich ein Raster gezeichnet ist, über dem sich die Graphen mehrerer Signale befinden. Zugleich ist ein Menü geöffnet, dessen Name direkt über der schwarzen Leiste angegeben ist. Das Menü besteht aus einigen Untermenüs, denen jeweils eine Spalte und ein Parameter zugeordnet wird. Die gewählten Parameter sind in der schwarzen Leiste im unteren Teil des Bildschirms aufgeführt. Wird ein Untermenü geöffnet, dann werden alle möglichen Optionen aufgelistet, wobei der gewählte Parameter dabei grün und die restlichen Werte rot gefärbt werden. Die Ebenen, aus denen dieser Bildschirm besteht, scheinen sich fälschlicherweise häufig zu überlagern, weshalb die in Abbildung 1 zu sehende Oberfläche nicht besonders repräsentativ für die tatsächliche Bedienung des Bildschirms ist, welche sich als sehr schwerfällig erweist. Die rechts des Bildschirms positionierten Tasten öffnen unterschiedliche Menüs, darunter das Triggermenü, das Kopplungsmenü und das Menü für Messungen. Nur Ersteres wurde bislang mit einer größeren Auswahl an Optionen ausgestattet, hat aber ebenfalls keinen Einfluss auf die Darstellung von Signalen. Die sechs Tasten, die unter dem Bilschirm liegen, dienen zur Navigation in den Menüs und können das zugehörige Untermenü öffnen oder in diesem den gewählten Parameter auf den darauffolgenden Wert setzen. Die Erkennung der Positionierung der Maus über den Tasten erfolgt dabei nicht zuverlässig und häufig sind Offsets der interpretierten Mausposition erkennbar. Dies findet sowohl beim Anklicken der Bildelemente statt, aber auch, wenn die Bildelemente durch einen breiteren Rahmen hervorgehoben werden, falls sich die Maus über ihnen befindet.

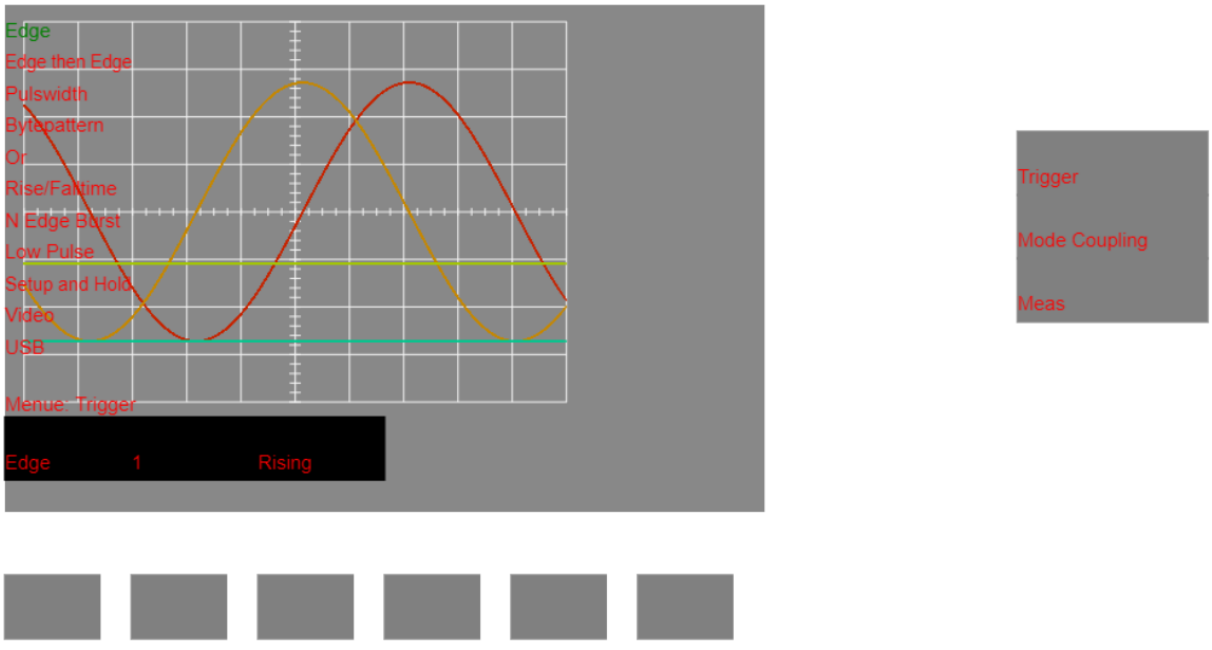

Abbildung 1: Ausgangszustand des virtuellen Oszilloskops

In Abbildung 2 ist die Testsektion zu sehen, die in der vorangegangenen Studienarbeit erstellt wurde, um gewisse Bedienelemente und Eingabefunktionen temporär zu ersetzen, solange sie noch nicht im virtuellen Oszilloskop umgesetzt sind [Vgl. 10, S. 16]. Ihre Werte fließen aber bereits in die Konfiguration des Oszilloskops ein. Im oberen Teil der Sektion sind einige Tasten angeordnet. Mit der ersten Taste lässt sich der Trigger aktivieren, während die zweite zwischen einer steigenden oder fallenden Flanke auswählt. Zudem ist der Kanal auswählbar, auf den getriggert wird und für die vier Kanäle sind jeweils ein grüner Kasten vorhanden, der sich bei Betätigung rot färbt und den entsprechenden Kanal deaktiviert. Darunter lässt sich der Grenzwert für den Trigger eingeben und die Frequenz und Amplitude einer Spannungsquelle vorgeben, welche zukünftig aber irrelevant sein wird.

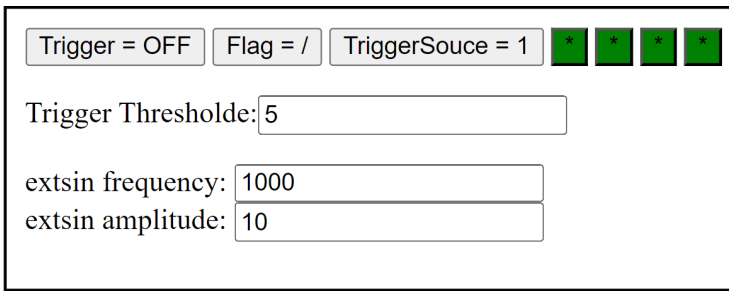

Abbildung 2: Testsektion zur Konfiguration des virtuellen Oszilloskops

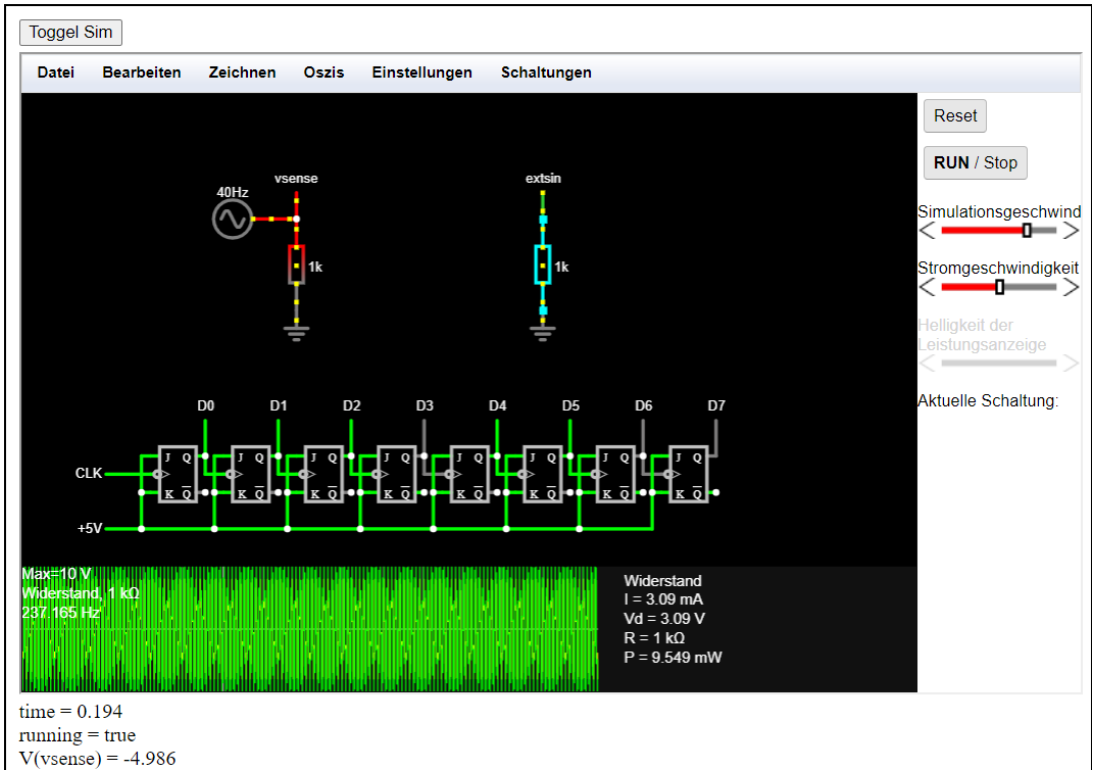

Abbildung 3: CircuitJS Simulation mit Toggle-Taste und Messwerten

Der Simulationsbereich verfügt über eine Toggle-Taste, über welchen sich das darunter befindliche, in Abschnitt 2.3 beschriebene CircuitJS Simulationsfenster ausblenden lässt [Vgl. 10, S. 15]. Unter der Simulation werden einige Simulationsdaten, wie die in der Simulation vergangene Zeit, und die an verschiedenen elektrischen Bauteilen anliegenden Ströme und Spannungen aufgelistet.

2.4.2 Programm

Die HTML-, CSS- und JavaScript-Dateien der Anwendung befinden sich im Ordner app, während die lokal gespeicherte CircuitJS-Applikation im Ordner war lokalisiert ist. Die Anwendung nutzt HTML5 und CSS3, bei denen es sich um die neuesten Versionen von HTML [Vgl. 2, S. 31] und CSS [Vgl. 2, S. 32] handelt. Die Version 2.7.3 des Schaltungssimulators wird in index.html als iframe-Element in die Anwendung integriert [Vgl. 10, S. 15]. Das HTML-Element iframe wird genutzt, „um externe HTML-Dokumente in das aktuelle HTML-Dokument einzubetten.“ [5, S. 250] Um einen Datenaustausch zwischen dem Simulator und der Anwendung zu gewährleisten, ist die lokale Ablage der Dateien des Simulators zwingend erforderlich. Das JavaScript-Programm des virtuellen Oszilloskops ist funktional in zahlreiche Dateien unterteilt, wie in der Ordnerstruktur der Anwendung in Abbildung 4 zu sehen ist. Diese starke Aufteilung macht die Navigation sehr kompliziert. Zudem finden einige dieser Dateien zum aktuellen Zeitpunkt keine Verwendung im Betrieb der Applikation.

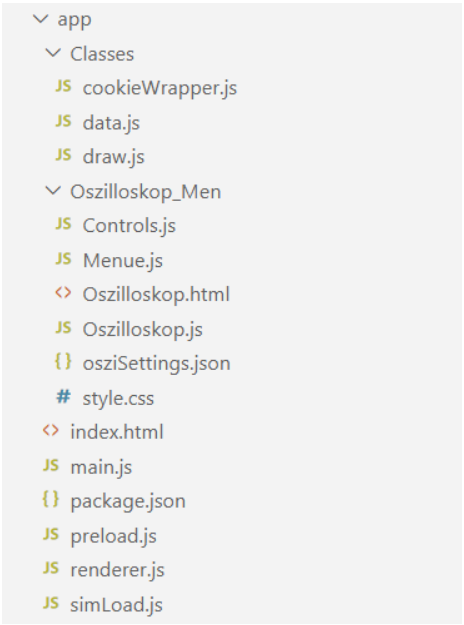

Abbildung 4: Ordnerstruktur der Webanwendung

Der JavaScript Code der Anwendung ist objektorientiert programmiert, das heißt die Strukturierung des Codes erfolgt in Form von Objekten, welche Gegenstände der Anwendung repräsentieren und die Funktionen der Anwendung durch gegenseitige Interaktion erfüllen [Vgl. 11, S. 701]. Der bislang bestehende Code umfasst zahlreiche Objekte für die Bildelemente [Vgl. 10, S. 11] und für die Steuerung des Menüverhaltens [Vgl. 10, S. 12]. Dabei sind mehrere Objekte als Instanzen von Klassen umgesetzt, obwohl sie die einzigen Objekte dieser Art sind. Dies ist z.B. beim Oszilloskopbildschirm osziScreen der Fall. Zudem werden einige abstrakte Klassen, von denen keine Instanzen erstellt werden, zur Unterbringung bestimmter Funktionen benutzt [Vgl. 10, S. 11]. Das Programm verfügt dementsprechend über sehr viele Klassendefinitionen, die teilweise nur eine geringe Sinnhaftigkeit besitzen.

Das Modell des virtuellen Oszilloskops wird auf einem Canvas-Element gezeichnet. Dieses stellt eine Zeichenfläche im HTML-Dokument dar, die mithilfe von JavaScript gefüllt werden kann [Vgl. 5, S. 237 f.]. Bei den in Abbildung 1 zu sehenden Graphen handelt es sich um in JavaScript definierte Signale, darunter zwei Sinus-Funktionen und zwei Konstantwerte. Demnach findet bislang keinerlei Übertragung der Messdaten aus der Simulation zum virtuellen Oszilloskop statt. Die vorliegende Umsetzung der Signalauswertung auf Trigger ist darüber hinaus nicht sehr effektiv gestaltet. Außerdem wurde in der vorangegangenen Studienarbeit versucht, die im Oszilloskopmenü getroffenen Einstellungen als Cookies zu speichern. Dieser Ansatz ist allerdings nicht funktionstüchtig und muss für dieses Projekt verworfen werden.

2.5 Vorgehensweise

Zur Editierung der Anwendung wird der Hauptordner in Visual Studio Code geöffnet und kann mithilfe des Plugins Live Server von Ritwick Dey als lokaler Webserver im Browser Google Chrome ausgeführt werden [Vgl. 12]. Zum Debugging konnten vom Programm Parameter in die Konsole übergeben und dort ausgelesen werden. Bei der Bearbeitung der Datenaufnahme, -verarbeitung und -ausgabe muss auf eine performante Auslegung geachtet werden, da sie in schnellen Zyklen stetig wiederholt werden. Bei der Umsetzung des Menüs ist es vorerst eher von Interesse, dass die bereits angelegten Optionen vollständig implementiert werden und nicht, dass die Menüauswahl vervollständigt wird.

3 Durchführung

Aus den genannten Gründen wurde die gesamte Anwendung umfassend überarbeitet und die neue Version des Programms wird im Folgenden beschrieben. Die Beschreibung der Anwendung ist dabei nach der funktionalen Bedeutung der Programmabschnitte gegliedert und fokussiert sich besonders auf die grundlegend veränderten und die neu hinzugekommenen Funktionen. Auch die Programmstruktur wurde deutlich modifiziert, indem alle Elemente, die keine Verwendung fanden, entfernt wurden und alle bisherigen JavaScript-Dateien zur Datei main.js zusammengefasst wurden. Daneben ist der Web Worker worker.js definiert, der in Kapitel 3.4 detailliert beschrieben wird. Die Navigation im Programm konnte damit deutlich erleichtert werden.

3.1 Erstellung der Bildelemente

3.1.1 Anlegen der Grafikobjekte

Die Oszilloskopkomponenten, die stets angezeigt werden, stellen die grundlegenden Bildelemente dar. Dazu gehören alle Knöpfe, aber auch der Oszilloskopbildschirm. Da er nur einmal vorkommt, wird er in Literal Schreibweise als Objekt deklariert, während die Knöpfe als Klasse definiert werden, von der für jeden angezeigten Knopf eine Instanz geschaffen werden muss. Das Objekt osziScreen, das den Bildschirm repräsentiert besitzt sowohl relative als auch absolute Werte für seine Koordinaten, seine Breite und Höhe. Diese Daten werden ebenfalls als Absolutwerte für das innere Rechteck gespeichert, auf dem später die Signale gezeichnet werden. Dabei dienen die relativen Maße dafür, dass sich die Position und Größe des Bildschirms proportional zur Fensterbreite ergeben und dabei das Seitenverhältnis der Bildelemente beibehalten wird. Damit die absoluten Werte einfach, ohne Berechnungen, abgerufen werden können, werden sie ebenfalls festgehalten. Außerdem verfügt osziScreen über die Attribute xSquare und ySquare, welche die x- und y-Skalierung der gezeichneten Signale als Zeitdauer bzw. als Spannungsdifferenz eines Quadrats in der Signalanzeige angeben. Dagegen halten xScale und yScale fest, welcher Zeit- und Spannungsintervall sich daraus für die gesamte Signalanzeige ergibt. Für die Vorgabe dieser Parameter durch den Nutzer werden neue Eingabefelder in index.html wie folgt definiert:

```
Time per square (s): <input id="tps" value="0.000125">
<br>
Voltage per square (V): <input id="vps" value="1">
```

Quellcode 1: Implementierung der Eingabefelder zur Skalierung

Für die Darstellung des Oszilloskopbildschirms muss beachtet werden, dass es sich bei dem Bildschirmhintergrund, den Graphen der visualisierten Messwerte und dem Menü gewissermaßen um eigene Ebenen handelt, die übereinander platzierbar sein müssen. Anstatt diese Ebenen sich auf einem Canvas ständig gegenseitig überschreiben zu lassen, ist es eine performantere Lösung, ein Canvas-Element für jede Ebene des Bildschirms zu erstellen. Dementsprechend werden in der HTML-Datei die drei Canvas Elemente background, graphs und menu deklariert und im div-Element oszi integriert, wie in Quellcode 2 zu sehen ist. Als div-Element unterteilt das Element oszi die Webseite und erzeugt einen neuen Bereich [Vgl. 3, S. 96].

```
<div id="oszi">
    <canvas id="background"></canvas>
    <canvas id="graphs"></canvas>
    <canvas id="menu"></canvas>
</div>
```

Quellcode 2: Anlegen der Canvas-Elemente für die Bildschirmebenen

Die Stylisierung der in Quellcode 2 befindlichen Elemente findet in style.css statt und ist in Quellcode 3 abgebildet. Dem div-Element oszi wird eine relative Positionierung vorgegeben, sodass es aus dem Dokumentenfluss herausgenommen wird, alle anderen Elemente sich aber so verhalten, als wäre es nie anderes positioniert gewesen [Vgl. 3, S. 127]. Dies äußert sich darin, dass oszi bei der Platzierung alle anderen Anwendungselemente aus dem abgedeckten Bereich verdrängt und es zu keiner Überschneidung kommt. Ein HTML-Element mit absoluter Positionierung wird ebenfalls aus dem Dokumentenfluss entfernt, kann aber andere Elemente überschneiden [Vgl. 3, S. 128], weshalb alle Canvas-Elemente mit absoluter Positionierung stylisiert werden. Ihre Positionierung erfolgt relativ zum Elternelement [Vgl. 3, S. 128], hier also relativ zum Bereich oszi. Das Element oszi erhält außerdem einen Abstand zum Fensterrand von zehn Pixeln und einen schwarzen, zwei Pixel breiten Rahmen. Die Canvas-Elemente werden zudem anfänglich alle mit den Eigenschaften top und left in der oberen linken Ecke von oszi positioniert. Die Canvas-Elemente unterscheiden sich lediglich in der Eigenschaft z-index, welche die Anordnung der Ebenen bestimmt. Umso höher der z-index Wert eines Elements ist, desto mehr wird es in den Vordergrund gesetzt [Vgl.

5, S. 467]. Der Canvas background bildet somit die hinterste Ebene, weshalb ihm zusätzlich ein weißer Hintergrund gegeben wird [Vgl. 5, S. 428]. Davor befinden sich erst die Graphen und im Vordergrund das Menü. Wird den Ebenen keine Hintergrundfarbe zugewiesen, dann sind sie transparent und die Inhalte der dahinterliegenden Ebenen sind sichtbar.

```
#oszi {
    position: relative;
    padding: 10px;
    border: 2px solid black;}
#background, #graphs, #menu {
    position: absolute;
    top: 0;
    left: 0;}
#background {
    background: "white";
    z-index: 0;}
#graphs {
    z-index: 1;}
#menu {
    z-index: 2;}
```

Quellcode 3: Stylisierung der Canvas-Elemente

Das Objekt osziScreen verfügt über die Methode setPos, welche die relativen Koordinaten und Maße auf die übergebenen Werte setzt und danach die Methode updatePos von osziScreen ausführt. Sie aktualisiert die Absolutwerte für die Positionierung des inneren und äußeren Rechtecks, welche die Darstellung der Signalanzeige und des Oszilloskopbildschirms bestimmen. Durch updatePos werden außerdem alle Inhalte in den beiden Canvas-Elementen graphs und menu entfernt und deren Koordinaten und Maße auf die des inneren bzw. äußeren Rechtecks des Bildschirms gesetzt. Zuletzt wird noch die Methode drawIncrements ausgeführt, die das Raster der Signalanzeige im Bildschirm zeichnet. Indem die Canvas-Elemente graphs und menu auf bestimmte Bereiche begrenzt werden, wird verhindert, dass Grafikelemente außerhalb dieses Bereichs gezeichnet werden. Die übrigen Methoden draw und drawChart dienen zum Zeichnen der Signale und werden in Abschnitt 3.2.2 genauer erläutert.

Bei den Bildelementen, aus denen sich das bislang bestehende Oszilloskopmodell zusammensetzt, handelt es sich neben dem Bildschirm ausschließlich um Tasten. Deshalb ist es möglich, diese als Instanzen der Klasse Button umzusetzen. Als Eigenschaften besitzen sie ebenfalls relative und absolute Positionsangaben. Außerdem erhalten sie einen Namen, der für die rechten Knöpfe der Beschriftung der Felder entspricht

und bei den unteren Tasten deren Nummer von eins bis sechs darstellt. Auch ihre absoluten Koordinaten und Maße werden durch die Methode updatePos als Produkt der relativen Positionsdaten und der Breite des Canvas background berechnet.

In updatePos wird zudem die Methode draw ausgeführt, die in Quellcode 4 zu sehen ist. Für den Hintergrund wird ein Rechteck mit den absoluten Positionsdaten des Knopfes grau gefüllt. Daraufhin wird ein schwarzes, einen Pixel breites Rechteck als Rahmen gezeichnet. Falls das Datenformat des Namen der Taste einem String entspricht, dann wird dieser Name als Beschriftung der Fläche abgebildet. Dazu wird die Schriftgröße und Schriftart, sowie die Farbe des Textes definiert. Des Weiteren wird der Text vertikal und horizontal zentriert und genau in der Mitte des Tastenfeldes platziert.

```
draw() {
    backgroundCtx.fillStyle = "grey";
    backgroundCtx.fillRect(this.xpoint,this.ypoint,this.width,this.height);
    backgroundCtx.strokeStyle = "black";
    backgroundCtx.lineWidth = 1;
    backgroundCtx.strokeRect(this.xpoint,this.ypoint,this.width,this.height);
    if(typeof this.name === 'string') {
        backgroundCtx.font = "15px Arial";
        backgroundCtx.fillStyle = "black";
        backgroundCtx.textAlign = "center";
        backgroundCtx.textBaseline = "middle";
        backgroundCtx.fillText(this.name,this.xpoint+this.width/2,
        this.ypoint+this.height/2,this.width);}}
```

Quellcode 4: Zeichnen der Tasten

Die Programmierung der Methode isInside ist in Quellcode 5 angegeben. Dabei erhält isInside die x- und y-Koordinaten des Mauszeigers und prüft, ob sich diese innerhalb der Fläche des jeweiligen Knopfes befinden. Falls dies der Fall ist, gibt die Methode einen positiven bool-Wert zurück und ansonsten einen negativen. Dies wird unter anderem dazu eingesetzt, bei der Bewegung der Maus festzustellen, ob eine der Tasten hervorzuheben ist. Für diesen Zweck dient die letzte Methode highlight, welche die Breite des Tastenrahmens zur Markierung auf zwei Pixel ausdehnt.

```
isInside(xmouse, ymouse) {
    const distanceX = this.xpoint + this.width;
    const distanceY = this.ypoint + this.height;
    if(xmouse >= this.xpoint && xmouse <= distanceX && ymouse >= this.ypoint
    && ymouse <= distanceY) {return true;} else {return false;}}
```

Quellcode 5: Methode isInside der Klasse Button

3.1.2 Initialisierung des virtuellen Oszilloskops

Beim Starten der Anwendung wird die Methode init ausgeführt, welche dem Objekt oszi untergeordnet wurde. Quellcode 6 stellt einen Ausschnitt aus der Deklaration von oszi dar. Dieses Objekt besitzt die Attribute simVisible, welches die Einblendung der Circuit-JS Simulation wiedergibt und dementsprechend boolsche Werte annimmt, sowie das Array elements. In das Array werden alle vorhandenen Tasten bei deren Erstellung hinzugefügt. Dies wird genutzt, damit einerseits bei der Änderung der Fenstergröße auch alle Tasten neu skaliert werden und andererseits bei der Bewegung der Maus für alle Tasten überprüft werden kann, ob sich der Mauszeiger über diesen befindet und sie hervorgehoben werden müssen.

Durch die Methode init wird die Höhe des HTML-Elements oszi, welches hier als div bezeichnet wird, auf einen Anteil dessen eigener Breite gesetzt. Dabei wird die Breite offsetWidth betrachtet, welche die Größe von Rahmen, Abstandszonen und dem Scrollbalken berücksichtigt [1]. Das Seitenverhältnis wurde passend zur Größe des bisherigen Stands des Oszilloskopmodells gewählt und muss gegebenenfalls angepasst werden, wenn das Modell weiterentwickelt wird. Dem Canvas-Element background werden die Maße von div übertragen. Dazu wird allerdings die Eigenschaften clientWidth und clientHeight genutzt, bei denen weder der Rahmen, noch der Scrollbalken einfließen [1]. Dadurch wird der Rahmen des HTML-Elements oszi nicht vom weißen Hintergrund des Canvas background überdeckt. Anschließend werden dem Oszilloskopbildschirm seine relativen Positionsdaten verliehen. Dabei wurde der Wert des letzten Parameters der Methode setPos derart gewählt, dass sich in Kombination mit den anderen Werten aus dem Raster der Signalanzeige Quadrate ergeben.

```
var oszi = {
    simVisible: false,
    elements: [],
    init: function() {
        div.style.height = 9/16 * div.offsetWidth + "px";
        backgroundCanvas.width = div.clientWidth;
        backgroundCanvas.height = div.clientHeight;
        osziScreen.setPos(0.02, 0.02, 0.6, 0.4571);
```

Quellcode 6: Objekt oszi und Initialisierung

Die Initialisierung muss außerdem die Erstellung aller Knöpfe als Instanzen der Klasse Button umfassen. Dazu werden dem Konstruktor die gewählten relativen Positionsdaten und der Name der Taste übergeben. Quellcode 7 zeigt als Beispiele die Generierung der ersten Taste zur Navigation im Menü und des Triggerknopfes.

```
this.button1 = new Button(0.05, 0.5, 0.03, 0.02, 1);
this.triggerButton = new Button(0.65, 0.05, 0.08, 0.04, "Trigger");
```

Quellcode 7: Beispiele für die Erstellung von Tasten

Ein Event-Handler ist eine Funktion in JavaScript, die bei einem bestimmten Ereignis ausgelöst [Vgl. 5, S. 792]. In Quellcode 8 wird das Objekt background mithilfe von addEventListener mit einem Event-Handler verknüpft [Vgl. 5, S. 794], der für das Ereignis mousemove aktiviert wird, wenn sich der Mauszeiger über dem Objekt bewegt [Vgl. 5, S. 798]. Indem die Funktion das beschriebene Array elements durchläuft, kann die entsprechende Taste hervorgehoben werden, wenn sich die Maus über diesem Bildelement befindet und die Methode isInside deshalb einen positiven Rückgabewert liefert. Andernfalls werden die Tasten regulär gezeichnet, damit sie gegebenenfalls vom hervorgehobenen in den Ausgangszustand zurückversetzt werden können. Die Koordinaten, die der Methode isInside übergeben werden, bestehen aus der pageX bzw. pageY Eigenschaft des Mausereignisses und einem Offset, der sich durch die in CSS definierte Abstandszone padding abzüglich der halben Rahmenbreite ergibt. Der Vorteil der Benutzung von event.pageY ist, dass sie die Mausposition relativ zum Rand des Dokuments bestimmt und deshalb auch Scrollbewegungen berücksichtigt werden [Vgl. 1].

```
backgroundCanvas.addEventListener("mousemove", (event) => {
    for(element of this.elements) {
    if(element.isInside(event.pageX - 9, event.pageY - 9)) {
        element.highlight();
        break;
```

Quellcode 8: Deklaration Event Listener bei Bewegung der Maus

Das Objekt window repräsentiert das Browserfenster, in welchem das Skript läuft [Vgl. 3, S. 271 f.]. Ihm kann ein Event Handler angehangen werden, der eine Funktion ausführt, wenn das resize Ereignis, also die Veränderung der Größe des Fensters, auftritt. Dies ist in Quellcode 9 zu sehen. In der entsprechenden Funktion werden dabei zuerst die gleichen Schritte vorgenommen wie bei der ursprünglichen Initialisierung in Quellcode 6, indem das Seitenverhältnis des div-Bereiches wieder hergestellt und das

Canvas-Element background an dessen Breite und Höhe angepasst wird. Zusätzlich werden hier aber alle aktuellen Grafikelemente auf background entfernt. Die absoluten Positionsdaten des Oszilloskopbildschirms und aller Tasten werden daraufhin mithilfe der jeweiligen updatePos Methode aktualisiert. Auch die Canvas-Elemente graphs und menu müssen als Reaktion auf die Veränderung der Fenstergröße angepasst werden, was in den kommenden Abschnitten noch aufgegriffen wird.

```
window.addEventListener("resize", (event) => {
    div.style.height = 9/16 * div.offsetWidth + "px";
    backgroundCtx.clearRect(0, 0, window.innerWidth, window.innerHeight);
    backgroundCanvas.width = div.clientWidth;
    backgroundCanvas.height = div.clientHeight;
    osziScreen.updatePos();
    for(element of this.elements) {
        element.updatePos();}
    osziSettings.updateMenu();
    osziScreen.draw(screenData);});
```

Quellcode 9: Deklaration Event Listener bei Größenänderung des Fensters

Als Resultat des in diesem Kapitel beschriebenen Skripts entsteht eine überarbeitete Oberfläche des virtuellen Oszilloskops, die in Abbildung 5 dargestellt ist. Dank der Event Listener passt sich das Oszilloskopmodell den Einflüssen des Bedieners an, indem es abhängig von der Fenstergröße skaliert wird und der Position des Mauszeigers entsprechend Bedienelemente hervorgehoben werden.

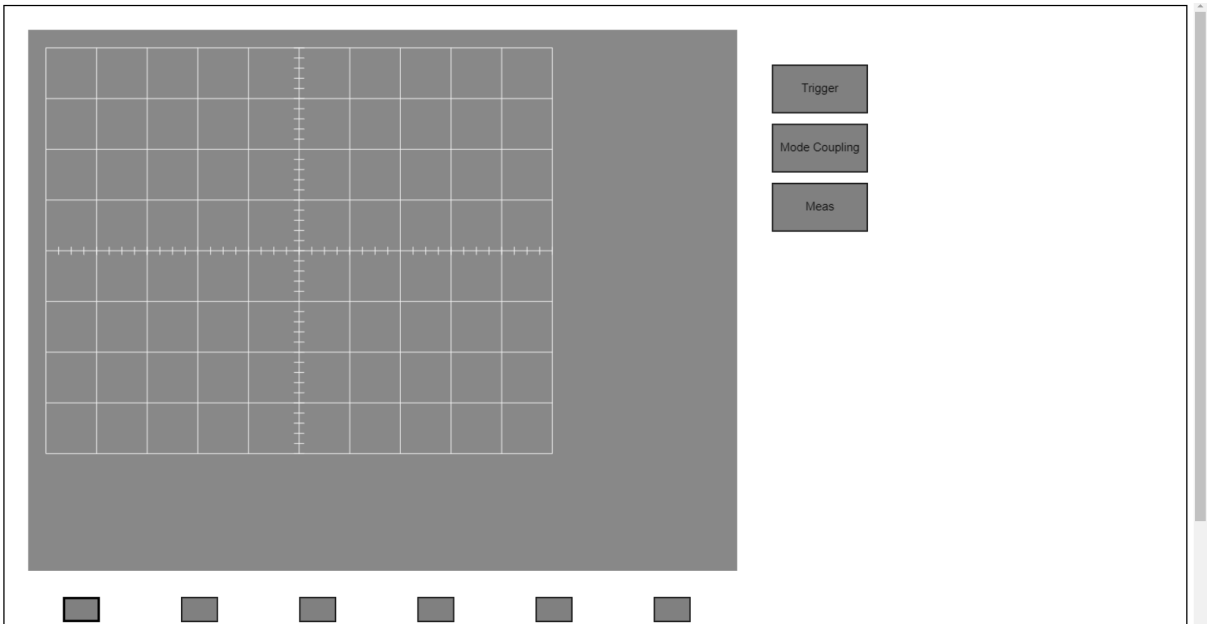

Abbildung 5: Überarbeitete Oberfläche des virtuellen Oszilloskops

3.2 Datenverarbeitung

3.2.1 Aufnahme und Speicherung von Messwerten

Die Aufnahme und Speicherung der aus der Simulation gewonnenen Messdaten bildet die Grundlage sowohl für das reguläre, dynamische Zeichnen der Signale auf der Oszilloskopanzeige, als auch für die Implementierung der Triggerfunktion. Zur Übertragung der Messwerte aus der Simulation in die restliche Anwendung wird die Funktion sim.getNodeVoltage eingesetzt, die das elektrische Potential, das an einem bestimmten Messpunkt vorliegt, wiedergibt. Dabei lässt sich der Messpunkt über dessen Namen wählen, welcher als Variable von getNodeVoltage eingesetzt werden muss. Bei sim handelt es sich um die das iframe-Element in der HTML und dementsprechend um die Simulation. Es wurde festgelegt, dass für die Datenaufnahme die Messpunkte CH1 bis CH4 für die vier Kanaleingänge erwartet werden. Zur Messwertübertragung müssen deshalb, in der Simulation, der Schaltung ein oder mehrere Benannte Knoten bzw. Labeled Nodes als neue Elemente hinzugefügt und ihnen die Namen CH1, CH2 usw. zugewiesen werden. Diese Komponenten lassen sich in CircuitJS unter (Zeichnen -> Ausgänge, Messgeräte, Text -> Benannten Knoten einfügen) auswählen [Vgl. 13].

Auf diese Weise ist es dafür möglich, die Messpunkte frei in der Schaltung zu platzieren. Der Nachteil an der Lösung der Datenübergabe per Benanntem Knoten ist, dass die Art und Bezeichnung der dafür verwendeten Schaltungskomponenten immer mit den angegebenen Vorgaben übereinstimmen müssen. Alternativ zu den Benannten Knoten umfasst CurcuitJS auch Voltmeter als Schaltungskomponenten, welche die Kanäle eines Oszilloskops symbolisieren. Für die Einbindung der Messdaten dieser Elemente ließ sich allerdings keine Möglichkeit ermitteln.

Zur Überprüfung der Implementierungen wurde ein gleichstromerregter RLC-Schwingkreises genutzt, da bei diesem sowohl Wechsel- als auch Gleichspannungen messbar sind. Bei der in Abbildung 6 zu sehende Beispielschaltung wurden zusätzlich die vier Benannten Knoten an verschiedenen Stellen der Schaltung positioniert. Über die CircuitJS Funktionalitäten war es zudem möglich, die Daten der Schaltung als txt-Datei zu speichern, sodass sie jederzeit erneut importiert werden kann.

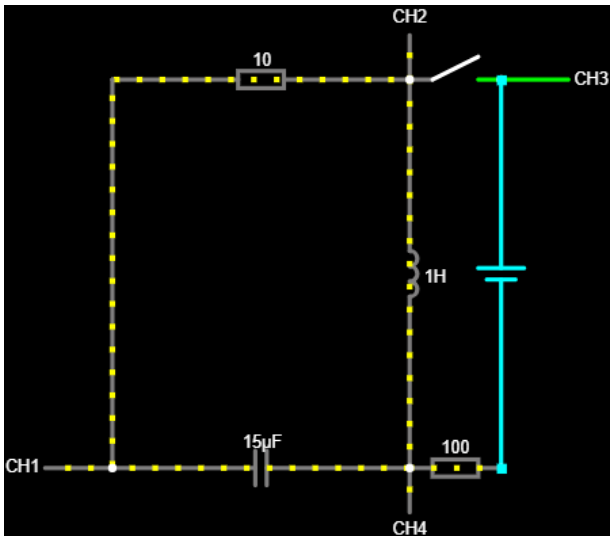

Abbildung 6: Beispielschaltung in CircuitJS

Für das Projekt ist vorgesehen, dass die Datenspeicherung in Form eines Ringspeichers zu erfolgen hat. Ringspeicher sind Datenpakete mit begrenzter Größe, bei denen neue Daten hinzugefügt werden, indem sie die ältestesten Elemente des Speichers überschreiben [Vgl. 11, S. 758]. Somit werden die Speicherplätze zyklisch wiederverwandt und der Speicher kann mit dem Kreislauf in einem Ring modelliert werden. Der Bedarf an mehreren dieser Speicher macht die Erstellung der Klasse Buffer erforderlich. Als Eigenschaften besitzt sie die Arrays CH1 bis CH4, die mit den Spannungswerten der Simulation gefüllt werden, sowie das Array timeSteps, das die Größe der Zeitschritte der Simulation zu den jeweiligen Messzeitpunkten festhält.

Die Funktionsweise des Ringspeichers lässt sich für die Eigenschaften von Buffer nachbilden, indem zuerst das erste Element der fünf Arrays mithilfe der Methode shift entfernt wird. Der Index aller anderen Elemente wird damit jeweils um eine Stelle verringert und die hinzuzufügenden Elemente können am Ende der Arrays integriert werden. Damit sich, nach dem Start der Anwendung, auf diese Weise auch ein leeres Array füllen kann, wird das neueste Element immer an der Stelle eingefügt, die der gewünschten Arraygröße entspricht. Die neuen Messwerte und Zeitschrittgrößen lassen sich mit den Funktionen sim.getNodeVoltage und sim.getTimeStep erhalten. Damit sich die Größe der Arrays anpassen lässt, wurde die globale Variable ringBufferSize erstellt, die der Anzahl von Messwerten entspricht, die gleichzeitig im Ringspeicher vorhanden sind. Vorerst wurde hierfür der Wert 2500 gewählt. In Quellcode 10 ist die Methode pushData angegeben, welche die eben beschriebenen Schritte durchführt. Die Bearbeitung der Kanäle zwei bis vier wurden allerdings gekürzt.

```
pushData() {
    this.CH1.shift();
    this.timeSteps.shift();
    this.CH1[ringBufferSize] = sim.getNodeVoltage("CH1");
    this.timeSteps[ringBufferSize] = sim.getTimeStep();}
```

Quellcode 10: Hinzufügen neuer Elemente in den Ringspeicher

Die zwei Objekte ringBuffer und screenData sind Instanzen der Klasse Buffer. Während in ringBuffer stetig die aktuellen Werte hinzugefügt und von diesem ausgehend gezeichnet werden, dient screenData als statischer Speicher, der immer die Daten von ringBuffer übernimmt, wenn ein Signal gezeichnet wird. Damit bleiben diese Daten erhalten und erlauben die Wiederherstellung der gezeichneten Graphen, nachdem diese durch die Veränderung der Fenstergröße gelöscht wurden. Das Objekt screenData stellt die Basis dar, um in Zukunft eine Navigation beim getriggerten oder gestoppten Signal durchzuführen. Durch die Hinterlegung des jeweiligen Zeitstempels für alle Spannungswerte kann deren Positionierung auf der Zeitachse von der Größe des vorangegangenen Zeitschritts abhängig gemacht werden. Gleichzeitig wäre auch die Veränderung der Skala, während die gezeichneten Signale beibehalten werden, durch die Umformung der Zeitachsenposition möglich.

Für die Klasse Buffer wurde zusätzlich die Methode getData zum Kopieren eines Datenbereichs aus einer anderen Buffer-Instanz definiert. Die Methode erhält als Argumente den gewünschten Start- und Endindex, welche den zu kopierenden Speicherbereich angeben. Da diese Methode vorerst nur für das Objekt screenData Verwendung findet, um Daten aus ringBuffer zu übernehmen, wird ringBuffer in diesem Fall konkret als Quelle definiert. Alle Eigenschaften von screenData werden somit auf die Werte von ringBuffer im definierten Speicherbereich gesetzt. Dabei wird die Methode slice genutzt, um die Daten zwischen den beiden vorgegebenen Indizes zu erhalten [Vgl. 1]. Die letzte Methode checkTrigger von Buffer ist Teil der Implementierung des Triggers und wird deshalb in Kapitel 3.4 erläutert.

3.2.2 Zeichnen von Datenpaketen

In Abbildung 7 ist das entwickelte Konzept visualisiert, wie der Ringspeicher und die darin enthaltenen Arrays durch die beiden Oszilloskopmodi des dynamischen Zeichnens und der Triggerfunktion genutzt werden kann. In der Grafik des Ringspeichers wandern die neuen Elemente im Laufe der Zeit im Uhrzeigersinn von oben rechts nach oben links, wo sie schließlich gelöscht werden. Damit die neuesten aufgenommenen Messdaten direkt im Oszilloskop dargestellt werden, befindet sich der Bereich, der für das dynamische Zeichnen als Grundlage dient, am Anfang des Ringspeichers. Die Überprüfung der Messwerte auf die Triggerbedingung muss dagegen nicht direkt erfolgen, weshalb sie mit den zentral im Ringspeicher gelegenen Elementen durchgeführt wird. Die Größe dieser Bereiche lassen sich über die globalen Variablen bundledCycles und triggerArea bestimmen.

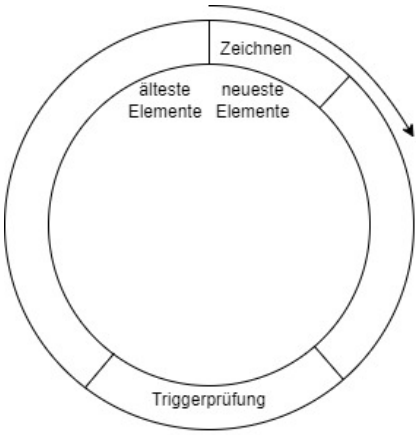

Abbildung 7: Nutzung der Daten im Ringspeicher

Die in Quellcode 11 vorliegende Funktion didStep wird für jeden Schritt der Simulation ausgeführt und regelt die Datenverarbeitung abhängig vom ausgewählten Zeichenmodus. In jedem Fall wird mit der in Abschnitt 3.2.1 erläuterten Methode pushData das neueste Datenpaket dem Ringspeicher ringBuffer hinzugefügt. Die globale Variable stepCounter dient als Zähler der durchgeführten Simulationsschritte und wird für jeden Zyklus von didStep um eins erhöht. Daraufhin wird zwischen den beiden Zeichenmodi des Oszilloskops unterschieden. Befindet es sich im Triggermodus, wird die Methode checkTrigger immer dann aufgerufen, wenn stepCounter ein Vielfaches der Variable bundledCycles beträgt. Der Wert von bundledCycles, der die Anzahl von Datenpaketen in der Anzeige bestimmt, wurde dabei mit 250 gewählt. Da der Ringspeicher 2500 Elemente umfasst, sind in ihm demzufolge die Daten von zehn aufeinanderfolgenden Bildern enthalten.

Im regulären Zeichenmodus dient dagegen die globale Variable drawTime als Indikator, die mit jedem Zyklus um die derzeitige Zeitschrittgröße der Simulation erhöht wird. Wenn sie das zehnfache der Eigenschaft xSquare und damit den Wert des Zeitintervalls überschreitet, der durch eine Bildschirmbreite repräsentiert ist, werden die Daten für den Zeichenbereich, der sich am Ende der Arrays befindet von ringBuffer auf screenData über die Methode getData kopiert. Das Speicherobjekt screenData kann daraufhin mithilfe der Methode draw von osziScreen abgebildet werden. Zusätzlich sind sowohl der Zähler stepCounter, sowie der Zeitintervall drawTime auf null zu setzen, damit sie für den nächsten Abbildungszyklus zur Verfügung stehen.

Solange die Eingabe für die horizontale Skalierung der Signalanzeige nicht null ist, wird für beide Zeichenfunktionen schließlich die Zeitschrittgröße der Simulation mit der Methode setTimeStep auf den Zeitintervall gesetzt, der sich als Quotient der Zeitdauer der gesamten Bildschirmbreite durch die vorgesehene Anzahl an Datenpaketen bundledCycles ergibt. Damit wird gewährleistet, dass die Anzahl der Datenelemente pro gezeichnetem Graph konstant und unabhängig von der horizontalen Skalierung ist. Für den Moment der Veränderung der horizontalen Skalierung kommt es aber dennoch zu Abweichungen. Im Hinblick auf die Funktion des Oszilloskops heißt dies, dass eine Anpassung der Auflösung der Datenaufnahme vorgenommen wird, ohne dass es zu Datenverlusten durch eine mögliche Filterung kommt. Allerdings ist die Methode setTimeStep fehlerbehaftet, sodass Warnungen in der Konsole ausgegeben werden, falls der kalkulierte Zeitschrittwert von dem Wert abweicht, der in den Einstellungen der Simulation festgelegt wird. Auch innerhalb der Simulation kommt es unter diesen Bedingungen zu Abweichungen.

```
function didStep() {
    ringBuffer.pushData();
    stepCounter++;
    if(osziSettings.triggerActivated) {
        if(stepCounter%bundledCycles == 0) {
            ringBuffer.checkTrigger();}
    } else {
        drawTime = drawTime + sim.getTimeStep();
        if(drawTime >= 10*osziScreen.xSquare) {
            screenData.getData(ringBufferSize - stepCounter, ringBufferSize);
            osziScreen.draw(screenData);
            stepCounter = 0;
            drawTime = 0;}}
    if(osziScreen.xSquare != 0) {
        sim.setTimeStep(10*osziScreen.xSquare / bundledCycles);}}
```

Quellcode 11: Funktion didStep zur Datenverarbeitung

Die bereits benannten Methoden draw und drawChart vom Bildschirm osziScreen sind in Quellcode 12 zu sehen. Durch draw wird zuerst der Speicher screenData dem übertragenen Speicherobjekt buffer gleichgesetzt. Da zum jetzigen Zeitpunkt ausschließlich screenData übergeben wird, ist dies vorerst nicht förderlich, kann aber verwendet werden, falls einmal andere Speicherelemente direkt gezeichnet werden. Danach muss das Canvas-Element graphs geleert und die Eingabewerte für die horizontale und vertikale Skalierung der Anzeige ausgelesen werden, die in den Eigenschaften xSquare und ySquare gespeichert sind. Mit der Funktion parseFloat wird dazu der String, den man durch die Ermittlung der Eigenschaft value der HTML-Elemente der Eingabefelder erhält, in eine Gleitkommazahl umgewandelt [Vgl. 1]. Die Eigenschaften xScale und yScale für die grafische Skalierung können durch die Teilung der Pixelanzahl der Breite bzw. Höhe der Anzeige durch den Zeit- bzw. Spannungsintervall, der auf dem Bildschirm abgebildet wird, berechnet werden. Die Elemente des Arrays channelData stellen die Daten von einem der Eingangskanäle dar und das Array channelColor umfasst die den Kanälen zugehörige Farbe der Graphen. Daraufhin lässt sich die Methode drawChart für alle Elemente von channelData ausführen, falls die entsprechenden Kanäle überhaupt aktiviert sind. Zuvor wird noch die Zeichenfarbe auf die definierte Farbe der Graphen gesetzt.

```
draw: function(buffer) {
    screenData = buffer;
    graphsCtx.clearRect(0, 0, this.width, this.height);
    this.xSquare = parseFloat(document.getElementById("tps").value);
    this.ySquare = parseFloat(document.getElementById("vps").value);
    this.xScale = this.innerWidth / (10 * this.xSquare);
    this.yScale = this.innerHeight / (8 * this.ySquare);
    let channelData = [buffer.CH1, buffer.CH2, buffer.CH3, buffer.CH4];
    let channelColor = ["#C82300", "#C88700", "#A5C800", "#00C88F"];
    for(var i = 0; i <= 3; i++) {
        if(osziSettings.activeChannels[i]) {
            graphsCtx.strokeStyle = channelColor[i];
            this.drawChart(channelData[i], buffer.timeSteps);}}},
drawChart: function(channelData, timeSteps) {
    graphsCtx.beginPath();
    graphsCtx.moveTo(0, this.innerHeight/2 - channelData[0]*this.yScale);
    let xCoord = 0;
    for(let i = 1; i < channelData.length; i++) {
        xCoord = xCoord + timeSteps[i];
        graphsCtx.lineTo(xCoord*this.xScale, this.innerHeight/2 -
        channelData[i]*this.yScale);}
```

Quellcode 12: Methoden draw und drawChart von Objekt osziScreen

Der Methode drawChart werden die Messwerte eines Kanals und das Array der zur selben Zeit erfolgten Zeitschritte übergeben. Sie beginnt an der linken Seite des Canvas graphs, eine Kurve zu zeichnen. Die Position der Spannungswerte wird immer relativ zur halben Höhe der Anzeige ermittelt, welche zum aktuellen Stand der Anwendung immer 0V repräsentiert. Ein Offset, der dies verändert, muss zukünftig implementiert werden. Innerhalb einer for-Schleife wird dann eine Linie entlang der restlichen, skalierten Positionen der Messwerte gespannt. Zur Positionierung auf der Zeitachse wird die Variable xCoord für jeden Zyklus um den entsprechenden Wert aus dem Array timeSteps inkrementiert. Mit stroke und closePath wird das Zeichnen der Linie schließlich ausgeführt und abgeschlossen [Vgl. 1].

In Abbildung 8 sind die gezeichneten Graphen für den beschriebenen RLC-Schwingkreis zu sehen. Entsprechend der Programmierung bildet die Anzeige eine Abfolge von Signalausschnitten ab. Im Gegensatz zum Bildschirm eines realen Oszilloskops wandern die Datenpunkte deshalb nicht von rechts nach links über den Bildschirm, was für die Anwendung zu leistungsintensiv gewesen wäre. Gegebenenfalls kann hierfür in einer kommenden Studienarbeit noch eine Lösung gefunden werden. Die allgemeine Visualisierung der Signalanzeige besitzt ebenfalls weiteren Optimierungsbedarf und für die Darstellung der Graphen stehen noch einige Konfigurationsoptionen aus.

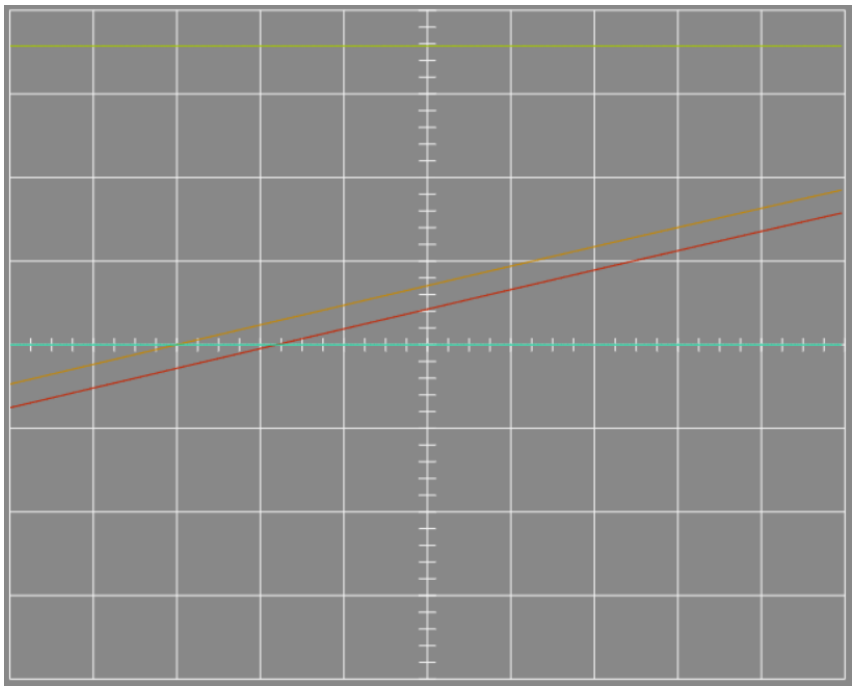

Abbildung 8: Signalanzeige mit gezeichneten Graphen

3.3 Umsetzung des Menüs

Für die Umsetzung des Menüs wird das Objekt osziSettings erstellt. Die Definition der Menüoptionen findet mit der Eigenschaft menu des Objekts statt. Durch die Methode setMenu, welche bei der Initialisierung der Anwendung ausgeführt wird, werden alle Untermenüs als Arrays in dem Array menu hinzugefügt. Ergänzend dazu besitzt osziSettings die Eigenschaften menuOptions und menuSettings, bei denen es sich ebenfalls um Arrays handelt. Während menuOptions wiedergibt, welche Untermenüs zum jeweiligen Zeitpunkt geöffnet sind, indem es die Indizes der Elemente in menu enthält, werden in menuSettings die getroffenen Einstellungen gespeichert, indem es für jedes Element von menu und damit für jedes Untermenü den Index der gerade ausgewählten Option angibt. Wenn die Auswahl der Optionen erweitert wird, bliebe diese Lösung zwar weiterhin performant, gleichzeitig macht es aber auch die Programmierung relativ unübersichtlich, weil zum Auslesen einer Option immer der Index der gewünschten Daten im Array menu bekannt sein muss. Die Eigenschaft activeMenu hält daneben die Beschriftung der zuletzt betätigten Menütaste und menuName den offiziellen Menünamen fest. Zusätzlich besitzt das Objekt mehrere Konfigurationen des virtuellen Oszilloskops, darunter triggerActivated für den Zustand des Triggers und TriggerThreshold für den ausgewählten Grenzwert des Triggers. Dazu gehören auch die Eigenschaft triggerPos, welche die horizontale Positionierung des Triggerereignis auf der Signalanzeige bestimmt, und das Array activeChannels, das boolsche Werte für die Zustände der Eingangskanäle beinhaltet.

In der Initialisierung des virtuellen Oszilloskops ist für die Erkennung der Betätigung von Tasten ein weiterer Event Handler hinzuzufügen. Dessen Funktion ist mit der von mousemove vergleichbar, er reagiert aber auf das Ereignis mouseup und damit auf das Loslassen der gedrückten Maustaste [Vgl. 5, S. 798]. Auch hier werden alle Bedienelemente durchlaufen und mit der Methode isInside überprüft, ob sich der Mauszeiger darüber befindet. Ist dies der Fall, führt die Funktion die Methode pressButton vom Objekt osziSettings aus und sendet als Parameter die Bezeichnung der ausgewählten Taste. Die Methode pressButton findet man in Quellcode 13 und ist für die Verwaltung des Menüs zentral, da sich dieses nur bei der Bedienung der Tasten verändert.

Wie in Abschnitt 3.1.1 beschrieben, werden die Tasten mit Beschriftung von den Tasten zur Navigation im Menü unterschieden. Bei ersteren stellt die Beschriftung den Namen des Elements dar, bei den anderen ist der Name eine Nummer. Dementsprechend wird auch in der Methode pressButton untersucht, ob es sich bei dem empfangenen Namen um einen String handelt. Trifft dies zu, wird geprüft, ob sich der empfangene Name mit

dem Namen des zuletzt geöffneten Menüs deckt. Nur wenn dies nicht der Fall ist und somit auf ein neues Menü zugegriffen wird, ist das aktuell angezeigte Menü und das Untermenü mit den Methoden closeMenu und closeSubmenu zu schließen. Die Eigenschaft activeMenu wird daraufhin auf den Namen des neuen Menüs gesetzt. Zudem ist eine switch-case-Verzweigung für den empfangenen Tastennamen definiert. Je nach Namen der Taste werden damit unterschiedliche Menüs geöffnet [Vgl. 1], für welche jeweils offizielle Bezeichnung und die Untermenüs vorzugeben sind, um sie mit der Methode openMenu zu öffnen. Trifft keine der case-Bedingungen zu, wird die Funktion unter default ausgeführt, wodurch eine Nachricht in der Konsole der Anwendung ausgegeben wird, dass der aktivierte Knopf nicht zugeordnet werden konnte. Ist das Datenformat des Tastennamens kein String, ist ebenfalls die Übereinstimmung des alten mit dem neuen Tastennamen zu überprüfen. Wird die gleiche Navigationstaste mehrfach hintereinander gedrückt, dann wird die Methode openSubmenu mit dem Argument true ausgeführt, sodass der gewählte Parameter auf die nächste Option in dem entsprechenden Untermenü springt. Andernfalls schließt die Methode closeSubmenu das aktuell geöffnete Untermenü und die Eigenschaft activeSubmenu wird mit dem um eins dekrementierten Wert von buttonName bestimmt. Das neue Untermenü wird daraufhin geöffnet, ein Sprung des Parameters findet aber nicht statt.

```
pressButton: function(buttonName) {
    if(typeof buttonName === 'string') {
        if(this.activeMenu != buttonName) {
            this.closeMenu();
            this.closeSubmenu();
            this.activeMenu = buttonName;
            switch(buttonName) {
                case "Trigger":
                    this.openMenu("Trigger Menu", [0, 1, 2]); break;
                case "Mode Coupling":
                    this.openMenu("Mode Coupling", [3]); break;
                case "Meas":
                    this.openMenu("Measurement", []); break;
                default:
                    console.log("Button " + buttonName + " not found!");}}}
    else {
        if(this.activeSubmenu == buttonName - 1) {this.openSubmenu(true);}
        else {
            if(this.activeSubmenu != null) {this.closeSubmenu();}
            this.activeSubmenu = buttonName - 1;
            this.openSubmenu(false);}}}
```

Quellcode 13: Methode pressButton zur Aktivierung von Tastenfunktionen

In den in Abschnitt 3.1.2 beschriebenen Event Handler für die Änderung der Fenstergröße wurde die Ausführung der Methode updateMenu von osziSettings integriert. Falls vor der Veränderung der Fenstergröße ein Menü oder Untermenü angezeigt wurde, öffnet die Methode diese wieder, nachdem sie zur Neuskalierung geschlossen wurden. Die Maße der Menüdarstellung sind dadurch immer an die Fenstergröße angepasst. Zur Vereinfachung des Auslesens der im Menü gewählten Parameter wird dem Objekt osziSettings die Methode getSetting aus Quellcode 14 angehängt. Dazu erhält die Methode den Index des gewünschten Untermenüs settingNumber. Die gewünschte Einstellung ist als String im Array menu im Element mit dem Index settingNumber gespeichert. Zur Ermittlung des Index der aktuellen Einstellung im jeweiligen Untermenü dient das Element in menuSettings mit demselben Index settingNumber.

```
getSetting: function(settingNumber) {
    return this.menu[settingNumber][this.menuSettings[settingNumber]];}
```

Quellcode 14: Methode getSetting zum Auslesen von Menüparametern

Gegenüber der Menüdarstellung in der Ausgangsversion der Anwendung befinden sich die Menüs und Untermenüs nun immer in schwarzen Boxen mit weißen Rahmen und die reguläre Farbe der Einträge ist weiß, wie in Abbildung 9 zu sehen ist. Die Positionierung der einzelnen Einträge wurde zudem angepasst. Für die Erweiterung des Menüs ist neben der Integrierung weiterer Menüs und Optionen auch erforderlich, dass das Menü deutlich dynamischer ausgelegt wird. Bei realen Oszilloskopen richtet sich beispielsweise die weitere Konfiguration des Triggers nach dem gewählten Triggermodus [Vgl. 7, S. 23], was mit dem jetzigen, statischen Oszilloskopmenü nicht möglich ist.

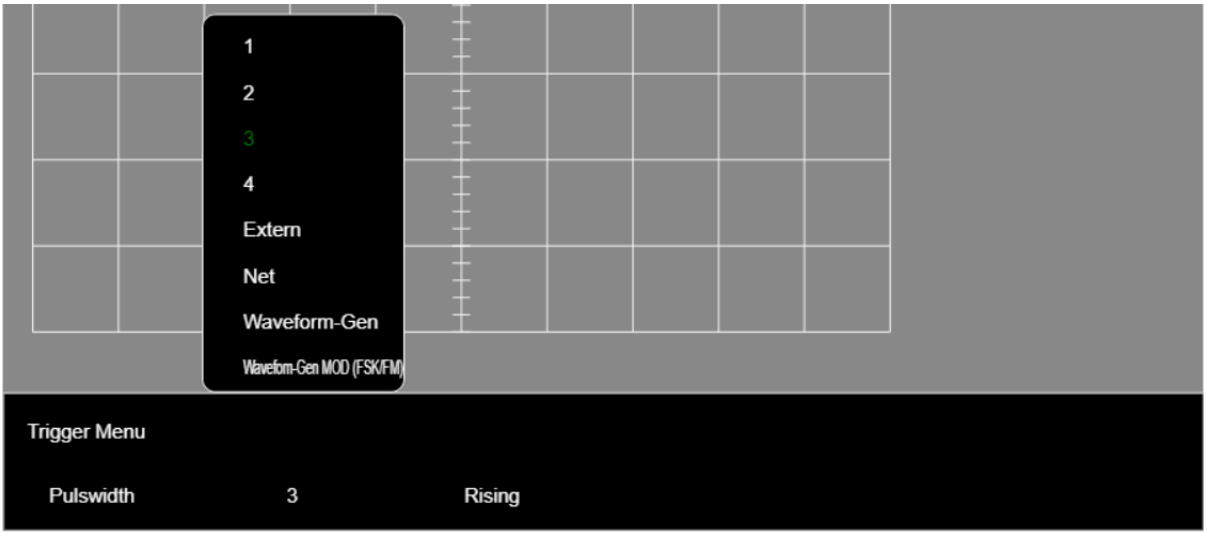

Abbildung 9: Überarbeitete Darstellung des Menüs

3.4 Implementierung der Triggerfunktion

Die Datenverarbeitung zur Erkennung der Triggerereignisse basiert auf dem in Abschnitt 3.2.1 erläuterten Ringspeicher. Gleichzeitig findet die Konfiguration des Triggers durch das in Kapitel 3.3 beschriebene Objekt osziSettings und dessen Eigenschaften statt. Das Zeichnen der getriggerten Signale erfolgt wiederum über die Methode draw von osziScreen. Für das Projekt ist die Auslagerung der Triggererkennung für die Messdaten in einen Web Worker festgelegt. Mit Web Workers lassen sich leistungsintensive, sequentielle Rechenprozesse durch Multi-Threading unterteilen und die Berechnungsdauer sich dadurch verkürzen. „Vereinfacht ausgedrückt, verwendet man den Begriff *Threading* oder *Multi-Threading*, wenn man ein Programm in mehrere Arbeiten bzw. Threads zerlegen will und diese dann quasi innerhalb des Programm parallel ablaufen. Eine echte parallele Ausführung wird mit mehreren Prozessorkernen möglich.“ [5, S. 889] Für den Web Worker muss eine separate JavaScript-Datei worker angelegt werden, auf die im HTML-Dokument verwiesen wird.

Wie in Abschnitt 3.2.2 dokumentiert, wird die Methode checkTrigger für das Objekt ringBuffer zyklisch ausgeführt. Die Methode ist in Quellcode 15 definiert. Die Variable triggerValues entspricht den Messdaten des ausgewählten Triggerkanals, von denen über die Methode slice ein Bereich kopiert wird, welcher den Abschnitt des Ringspeichers darstellt, der nach dem Nutzungskonzept aus Abschnitt 3.2.2 für die Triggerprüfung vorgesehen ist und dessen Größe durch die globale Variable triggerArea bestimmbar ist. Der Wert von triggerArea darf nicht kleiner als bundledCycles sein, da sonst ein Teil des Speichers nicht der Triggererkennung unterliegen würde. Je nach ausgewähltem Triggermodus kann es dagegen sinnvoll sein, den Bereich größer als bundledCycles zu wählen, damit auch Ereignisse an den Übergängen der Ausschnitte erfasst werden. Falls dadurch auf dasselbe Ereignisse mehrfach getriggert wird, ist das unproblematisch, da sich für beide Male identische Graphen ergeben. Die Funktion eval kommt dabei zum Einsatz, um den zusammengesetzten String als Quellcode zu interpretieren [Vgl. 1]. Der Triggermodus kann als erstes Element des Arrays menu mit getSetting ermittelt werden. Die Methode updateTrigger von osziSettings aktualisiert die Eigenschaften triggerThreshold und triggerPos, bei denen es sich um den Grenzwert des Triggers und die Positionierung des Triggerereignisses auf der Breite der Signalanzeige handelt, indem die Werte aus den Eingabefeldern im HTML-Dokument ausgelesen werden. Dem Web Worker werden daraufhin das Array der Messdaten des Triggerkanals, der Triggermodus, die gewählte Flag, der Grenzwert, sowie der Zähler für die Simulationsschritte mit der Methode postMessage übergeben [Vgl. 5, S. 893].

```
checkTrigger() {
    let triggerValues = eval("this.CH"+osziSettings.getSetting(1)+".slice
    (0.5*(ringBufferSize-triggerArea),0.5*(ringBufferSize+triggerArea))");
    let triggerMode = osziSettings.getSetting(0);
    osziSettings.updateTrigger();
    webworker.postMessage([triggerValues, triggerMode,
    osziSettings.getSetting(2), osziSettings.triggerThreshold, stepCounter]);}
```

Quellcode 15: Datenübertragung an den Web Worker zur Triggererkennung

Die Aktivierung des Web Workers erfolgt in der Methode toggleTrigger von osziSettings, die immer bei Betätigung der Triggertaste im Testbereich ausgeführt wird. Wie in Quellcode 16 definiert, wird zunächst die Eigenschaft triggerActivated von osziSettings invertiert. Falls der boolsche Wert daraufhin wahr ist, wird die Beschriftung der Triggertaste angepasst und die globale Variable webworker auf den neu erstellten Web Worker gesetzt, dessen Funktion in der Datei worker.js deklariert ist. Außerdem wird dem Objekt webworker ein Event Handler angehängt, der beim Empfangen einer Nachricht auslöst [Vgl. 1]. In der Funktion des Event Handlers lassen sich alle Eigenschaften des Speichers ringBuffer auf das Objekt triggerBuffer übertragen. Dabei wird Object.assign genutzt, damit triggerBuffer nicht einfach ringBuffer referenziert, sondern es sich weiterhin um zwei separate Objekte handelt, die individuell angepasst werden können. Der Speicher triggerBuffer wird somit als Kopie des momentanen Ringspeichers erstellt, sobald das Triggerereignis erkannt wird. Dieser Speicher kann in Zukunft als Basis für die Navigation im getriggerten Signal verwendet werden, kommt aber in diesem Projekt nicht weiter zum Einsatz. Die Variable leftBoundary repräsentiert die linke Grenze des Bereichs, die abhängig von den Daten der erhaltenen Nachricht, der Größe des Ringspeichers und des Triggerbereichs, dem aktuelle Zählerwert der Simulationsschritte, der Positionierung des Triggerereignisses und der Anzahl der Messwerte pro Signalanzeige kalkuliert wird. Der Web Worker liefert den Index des erkannten Triggerereignis im erhaltenen Array der Messdaten des Triggerkanals. Während dieser Berechnungsprozess stattfindet, laufen allerdings bereits neue Simulationsschritte ab, wodurch es zu einer Differenz zwischen dem ermittelten und dem tatsächlichen Triggerzeitpunkt kommt. Um dies zu umgehen, wird dem Web Worker der Schrittzähler zum Ausgangszeitpunkt übergeben und mit dem ermittelten Index addiert. Im Hauptprogramm wird bei Rückgabe dieser Summe der aktualisierte Zähler abgezogen und damit die Differenz ausgeglichen. Schließlich wird der Bereich zwischen der linken Begrenzung leftBoundary und dem rechten Ende, welches als Summe von leftBoundary und triggerArea bestimmt werden kann, aus dem Ringspeicher in screenData kopiert. Schließlich wird screenData gezeichnet.

Falls der Trigger durch die Betätigung des Knopfes deaktiviert wird, dann passt die Funktion die Beschriftung der Taste entsprechend an. Mit der Methode terminate lässt sich der Web Worker vom Hauptprogramm aus beenden [Vgl. 5, S. 896]. Zudem werden die beiden Zähler stepCounter und drawTime zurückgesetzt, damit sie für den regulären Zeichenmodus zur Verfügung stehen.

```
toggleTrigger: function() {
   this.triggerActivated = !this.triggerActivated;
   if(this.triggerActivated) {
      document.getElementById("Trigger").textContent = "Trigger: ON";
      webworker = new Worker("worker.js");
      webworker.addEventListener("message", function(messageEvent) {
         triggerBuffer = Object.assign({}, ringBuffer);
         let leftBoundary = messageEvent.data+0.5*(ringBufferSize-triggerArea)
         - stepCounter - osziSettings.triggerPos*bundledCycles;
         screenData.getData(leftBoundary, leftBoundary + triggerArea);
         osziScreen.draw(screenData);
      });}
   else {
      document.getElementById("Trigger").textContent = "Trigger: OFF";
      webworker.terminate();
      stepCounter = 0;
      drawTime = 0;}}
```

Quellcode 16: Aktivierung des Web Workers

Um den Triggermodus und die Flag verstellbar zu machen und gleichzeitig mit der Triggererkennung die bestmögliche Performanz zu erreichen, werden im Web Worker für die Umsetzung dieser Aufgaben Funktionszeiger verwendet. Die Variablen triggerFunc und flagFunc referenzieren demnach immer die Funktionen für die Datenauswertung entsprechend dem aktuell gewählten Triggermodus bzw. Flag. Sie werden nur verändert, falls der Triggermodus oder die Flag, welche in den empfangenen Daten enthalten ist, von der bislang gewählten Konfiguration abweicht. In diesem Fall werden die neuen Funktionen über getTriggerMode oder getFlag referenziert. Diese Funktionen weisen den Funktionszeigern über switch-case-Verzweigungen die passende Funktion zu. Dieser Vorgang wird äquivalent zum Hauptprogramm immer dann ausgeführt, wenn das Empfangen einer Nachricht den Event Handler auslöst. Daraufhin werden zusätzlich die Variablen data, triggerMode, triggerFlag, triggerThreshold und startIndex in der gleichen Reihenfolge aus der vom Hauptprogramm erhaltenen Nachricht ausgelesen, wie sie dort definiert worden sind. Der Funktionszeiger triggerFunc wird zudem ausgeführt.

```
self.addEventListener("message", function(messageEvent) {
    if(messageEvent.data[1] != triggerMode) {
        getTriggerFunc(messageEvent.data[1]);}
    if(messageEvent.data[2] != triggerFlag) {
        getFlagFunc(messageEvent.data[2]);}
    [data, triggerMode, triggerFlag, triggerThreshold, startIndex] =
    messageEvent.data;
    triggerFunc();});
```

Quellcode 17: Datenverwaltung im Web Worker

Infolge des Projekts wurden die Triggermodi Edge und Edge then Edge, sowie die Flags rising, falling, alternating und either implementiert, für die es jeweils eine Funktion gibt. Die Triggererfassung mit dem Triggermodus Edge ist in Quellcode 18 zu sehen. Dabei werden in einer for-Schleife die Messwerte durchlaufen und für jedes aufeinanderfolgende Wertepaar mit dem Funktionszeiger flagFunc überprüft, ob der definierte Grenzwert mit der vorgegebenen Flanke überschritten wurde. Falls dies zutrifft, wird startIndex um den Index des Triggerpunktes im Messwerte-Array inkrementiert und über postMessage an das Hauptprogramm zurückgegeben. Der bereits überprüfte Teil des Messwerte-Arrays kann daraufhin entfernt und die Funktion edgeTrigger erneut ausgeführt werden, um möglicherweise noch weitere Triggerpunkte in dem aktuellen Datenausschnitt zu ermitteln. Die Funktion edgeThenEdgeTrigger ist mit edgeTrigger sehr vergleichbar, hier wird aber eine im Web Worker definierte Variable secondEdge für jedes erkannte Triggerereignis invertiert und der Triggerindex dadurch nur bei jeder zweiten Erkennung zurückgegeben.

```
function edgeTrigger() {
    for(var i = 0; i < data.length; i++) {
        if(flagFunc(data[i], data[i + 1])) {
            startIndex = startIndex + i;
            self.postMessage(startIndex);
            data.splice(0, i + 1);
            edgeTrigger();
            break;}}}
```

Quellcode 18: Triggererkennung im Triggermodus Edge

Von den Funktionen, die vom Funktionszeiger flagFunc referenziert werden können, ist die Funktion risingFlag in Quellcode 19 als Beispiel gegeben. Sie erhält die zwei aufeinanderfolgenden Messwerte. Unter der Bedingung, dass der erste Wert unter dem festgelegten Grenzwert, der zweite aber darüber liegt, gibt die Funktion einen positiven boolschen Wert zurück. Ansonsten folgt false als Antwort. Für die fallende Flanke

müssen im Vergleich zu risingFlag lediglich die Vergleichsoperatoren invertiert werden. Daneben bietet das Oszilloskop von Keysight Technologies auch die Optionen alternating und either als Flag-Einstellung für den Triggermodus Edge und Edge then Edge an [Vgl. 7, S. 30], die bei alternierenden Flanken bzw. sowohl bei steigenden als auch fallenden Flanken auslösen. Bei der Funktion alternatingFlag wird mithilfe der Variable alternating zwischen einer steigenden und einer fallenden Flanke als Bedingung gewechselt, während bei eitherFlag beide Bedingungen mit dem Oder-Operator verknüpt werden. Damit sind alle möglichen Flags für die Triggermodi Edge und Edge then Edge implementiert. Für andere Triggermodi wie Pulse Width oder Rise/Fall Time [Vgl. 7, S. 21] sind allerdings in Zukunft noch weitere Konfigurationsoptionen umzusetzen.

```
function risingFlag(data_1, data_2) {
    if(data_1 < triggerThreshold && data_2 >= triggerThreshold) {
        return true;}
    return false;}
```

Quellcode 19: Triggererkennung für steigende Flanken

Die in Abschnitt 3.2.1 beschriebene Schaltung wird für die Überprüfung der Triggererkennung herangezogen. Der Trigger wird dabei auf den Modus Edge für steigende Flanken im Grenzwert 2V gesetzt und es wird auf den ersten Eingangskanal getriggert. Im Eingabefeld für die Positionierung des Triggers wird 0,5 angegeben, weshalb sich das Triggerereignis mittig im Bildschirm befinden müsste. Die Abbildung 10 zeigt die resultierende Signalanzeige. Es ist zu erkennen, dass der rote Graph, welcher den Kanal 1 darstellt, genau in der vorgesehenen Position den Grenzwert schneidet. Auch für andere Triggerkonfigurationen konnte die Implementierung erfolgreich überprüft werden.

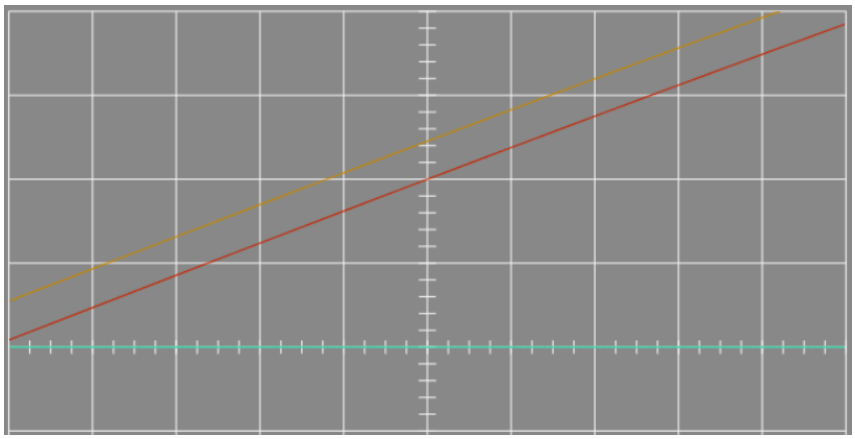

Abbildung 10: Beispielhafte Triggeranzeige

Der Web Worker zur Triggererkennung ist sehr performant ausgelegt und kommt erst bei der maximalen Simulationsgeschwindigkeit an seine Grenzen, bei der die Reihenfolge der vom Web Worker kommenden Daten durcheinander gerät, was teils dazu führt, dass nicht der richtige Abschnitt der Daten gezeichnet wird. Bei den noch ausstehenden Triggermodi ist deshalb auf eine sehr performante Umsetzung zu achten, besonders da diese auf teils noch komplexeren Prüfungen basieren als die beiden hinzugefügten Triggerarten. Zusätzlich sind die Optionen des Triggermenüs vom gewählten Triggermodus abhängig, was sich bisher nicht bemerkbar machte, weil der Edge und der Edge then Edge Modus die gleichen Optionen bieten. Beim Triggermodus Pulswidth kann beispielsweise vorgegeben werden, wie die Pulsbreite durch das Oszilloskop zu messen ist [Vgl. 7, S. 30]. Somit ist ein deutlich dynamischeres Menü als das jetzige für die vollständige Integration der ausstehenden Triggermodi erforderlich. Einige dieser Modi erfordern zudem die Übertragung der Zeitdaten in Form des Arrays timeSteps an den Web Worker.

3.5 Weitere Schritte

Der Anzeigename der Webanwendung, mit welcher der entsprechende Tab bezeichnet wird, konnte auf ‚Virtual Oscilloscope' gesetzt werden, indem der Name im head Bereich des HTML-Dokuments mit dem Element title definiert wurde [Vgl. 3, S. 59]. Zudem ließen sich die lokalen Dateien von CircuitJS, welche zur Einbindung des Simulators dienen, von der Version 2.7.3. auf 2.7.6 aktualisieren. Dazu wurde die Applikation als JavaScript Interface auf der Website von Paul Falstad [13] heruntergeladen und daraus das Verzeichnis war in den Ordner der Anwendung übernommen.

Neben Google Chrome wurde die Anwendung auch in den Browsern Edge und Firefox getestet. Während sie in Chrome und Edge problemlos läuft, fiel in Firefox auf, dass die Untermenüs nicht erscheinen. Dieser Fehler lässt sich darauf zurückführen, dass die Methode roundRect, welche für den Hintergrund der Untermenüs verwendet wird, nicht mit Firefox kompatibel ist [Vgl. 1]. Die Konfiguration des Oszilloskops lässt sich trotz dieses Fehlers wie gewünscht vornehmen, da die ausgewählten Parameter in der unteren Menüleiste angezeigt werden. In der Datei LICENSE wurden schließlich die Namen der Autoren aktualisiert.

4 Fazit

Die Anwendung des virtuellen Oszilloskops wurde von Grund auf überarbeitet. Zum einen wurde das über viele Dateien verteilte JavaScript-Programm zusammengeführt und modifiziert, wodurch einige Fehler behoben werden konnten. Durch diesen Vorgang ließen sich rund 2000 Zeilen JavaScript-Code in 700 Zeilen umwandeln. Zum anderen konnte die Anwendung um mehrere Funktionen erweitert werden. Dazu zählen die Speicherung der vorgenommenen Konfiguration des Oszilloskops, die Übertragung der Messdaten aus der Simulation in das virtuelle Oszilloskops und das Ablegen dieser Daten in einem Ringspeicher. Außerdem wurde die Triggerfunktion zum Teil umgesetzt und die eingebundene CircuitJS Applikation aktualisiert. Schließlich konnte ihre Funktionalität in den Browsern Google Chrome, Edge und Firefox erfolgreich geprüft werden, in Firefox kommt es allerdings zu Darstellungsfehlern.

In Bezug auf die umgesetzten Funktionen gibt es noch einige Verbesserungsmöglichkeiten. So findet bei der Datenverarbeitung aktuell keine Erkennung statt, ob für einen Kanal überhaupt ein Benannter Knoten existiert. Stattdessen werden für die Kanäle ohne Messpunkt konstant 0V im Oszilloskop angezeigt. Eine Unterscheidung im Zeichenprozess zwischen null und 0 als Messwert wäre hier sinnvoll. Eine mögliche Lösung für die abweichende Reihenfolge bei der Rückgabe der Triggerindizes vom Web Worker wäre die Ausnutzung der timeStep-Werte, die für die Implementierung kommender Triggermodi ohnehin dem Web Worker zu übertragen sind. Zur korrekten Darstellung der Anwendung in Firefox müsste die Verwendung von roundRect für die Untermenüs vermieden werden. Solange der Fehler von setTimeStep im Simulator besteht, ist die Größe der Zeitschritte jedes Mal auf 0 zu setzen, wenn die Anwendung gestartet wird. Nur so können die unerwünschten Ausgaben in der Konsole vermieden werden. Ist der Fehler behoben, dann wird dieser Schritt nicht mehr nötig sein und die Regelung der Zeitschrittgröße funktioniert wie gewünscht.

Um vom Startzustand der Anwendung ausgehend die Signale einer beliebigen Schaltung sehen zu können, muss die Simulation über die Toggle Sim Taste eingeblendet werden. Danach muss die gewählte Schaltung im Simulator erstellt und der Schaltung an den gewünschten Messpunkten Benannte Knoten (unter Zeichnen -> Ausgänge, Messgeräte, Text) hinzugefügt werden, die mit CH1 bis CH4 zu bezeichnen sind. Die Eingangskanäle sind daraufhin über die Kanaltasten zu aktivieren, woraufhin sich die dynamisch gezeichneten Signale auf dem Bildschirm zeigen. Gegebenenfalls muss zudem noch die Auflösung der Signalanzeige angepasst werden.

Im Vergleich zur Ausgangsversion läuft die Anwendung nun wahrnehmbar schneller und die Anwendungsdateien sind deutlich übersichtlicher. Es konnte dementsprechend auch eine gute Grundlage für kommende Projekte und Erweiterungen geschaffen werden. Wie in der Projektbeschreibung erläutert, ist es vorgesehen, dass die finale Anwendung eines Tages im Grundlagenlabor Elektrotechnik eingesetzt wird. Bis zur Finalisierung des virtuellen Oszilloskops müssen noch eine Vielzahl von physischen und funktionalen Komponenten eines realen Oszilloskops implementiert werden. In Abbildung 11 ist ein Oszilloskop der InfiniiVision 3000 X-Serie von Keysight Technologies zu sehen, welches dem virtuellen Oszilloskop als Vorlage dient. Von dieser großen Menge an Bedienelementen und den damit verbundenen Funktionalitäten ist offensichtlich nur ein kleiner Anteil umgesetzt. Zur Integration von Drehreglern als zweite Art von Bedienelementen neben den Tasten wäre die Erstellung einer abstrahierten Klasse für alle Bedienelemente denkbar, von der die Klassen für Tasten und Drehregler abgeleitet werden können. Letztendlich wäre es erstrebenswert, mit der grafischen Darstellung des virtuellen Oszilloskops die Gestalt des realen Oszilloskops aus Abbildung 11 möglichst detailgetreu nachzubilden, damit das im Grundlagenlabor vermittelte Wissen auch für den Umgang mit echten Oszilloskopen eingesetzt werden kann. Dadurch könnte die Anwendung auch optisch ansprechender gemacht werden. Zum jetzigen Zeitpunkt stimmt beispielsweise die Farbgebung der Graphen in der Anwendung noch nicht mit den tatsächlichen Farben der Eingangskanäle überein.

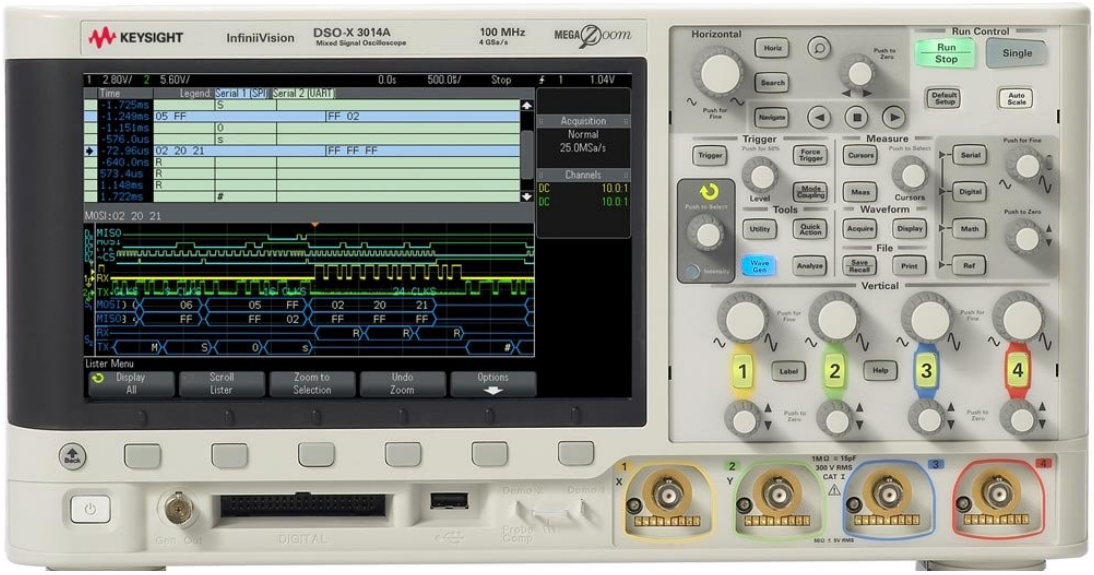

Abbildung 11: Oszilloskop Keysight InfiniiVision DSOX3000A [7]

Ein auf den umgesetzten Zeichenmodi aufbauender Erweiterungsschritt wäre die Implementierung einer Navigation im gestoppten, regulären Zeichenmodus und im getriggerten Signal. Für letzteres bietet sich der Speicher triggerBuffer als Basis der Daten an, ansonsten ist der Ringspeicher dafür geeignet. Unter diesen Bedingungen wäre das Skalieren der Graphen und das Betrachten von Messdaten, die sich vor oder hinter dem ursprünglich gezeichneten Graphen befinden, möglich. Die Navigationsfunktion würde die Messdaten dann in Abhängigkeit von den Zeitinformationen auf dem Bildschirm positionieren. Für die Nutzung dieser Funktion sind einige zusätzliche Bedienelemente nötig. Eine Auslagerung des Zeichnens der Signalgraphen in einen weiteren Web Worker wäre vorstellbar, da auf diese Weise das Hauptprogramm von viel erforderlicher Rechenleistung befreit werden könnte und die Laufzeit dieses Web Workers auch nebensächlich wäre, weil es keinen Rückgabewert gibt.

Die Umsetzung der restlichen Triggermodi mit den zugehörigen Konfigurationen würde an die Schritte in Kapitel 3.4 anknüpfen. Die Aktualisierung der Anwendung auf die neueste Version von CircuitJS, wie es in Kapitel 3.5 beschrieben wurde, muss in Zukunft regelmäßig vorgenommen werden, insbesondere damit die noch ausstehende Korrektur von setTimeStep in das virtuelle Oszilloskop übernommen wird. Es zeigt sich, dass noch zahlreiche Erweiterungen am virtuellen Oszilloskop ausstehen, bevor es alle Funktionen des realen Oszilloskops abdeckt und im Grundlagenlabor Elektrotechnik eingesetzt werden kann. Die Grundfunktionen eines Oszilloskop sind aber inzwischen vorhanden und es wurden bereits Maßnahmen getroffen, um die nächsten Erweiterungsschritte zu ermöglichen.

Literaturverzeichnis

[1] Open Web Docs. „MDN Web Docs“. (o.J.), Adresse: https://developer.mozilla.org/en-US/ (besucht am 14. 12. 2022).

[2] Martin Hahn, *Webdesign, Das Handbuch zur Webgestaltung* (Rheinwerk Design), 2. aktualisierte Auflage. Bonn: Rheinwerk Verlag, 2017.

[3] Jens Jacobsen, Matthias Gidda, *Webseiten erstellen für Einsteiger, Schritt für Schritt zur eigenen Website* (Rheinwerk Computing), 3. aktualisierte und erweiterte Auflage. Bonn: Rheinwerk Verlag, 2020.

[4] Manuela Hoffmann, *Modernes Webdesign, Gestaltungsprinzipien, Webstandards, Praxis* (Galileo Design Edition Page), 3. Auflage. Bonn: Galileo Press, 2013.

[5] Jürgen Wolf, *HTML und CSS, Das umfassende Handbuch*, 4. aktualisierte und überarbeitete Auflage. Bonn: Rheinwerk Verlag, 2021.

[6] Monika Budde. „Vorteile und Grenzen von CSS“. (2005), Adresse: http://page.mi.fu-berlin.de/mbudde/css_kurs/vorteile.html (besucht am 11. 03. 2023).

[7] Meilhaus Electronic GmbH. „Keysight InfiniiVision DSOX3000A Serie 2-/4-Kanal Oszilloskope bis 1GHz“. (o.J.), Adresse: https://www.meilhaus.de/dsox3000a.htm/ (besucht am 08. 03. 2023).

[8] Kevin Berisso, *An introduction to the Circuit Simulator Applet, for use in teaching relays and relay logic*. Memphis, 2018.

[9] Herbert Bernstein, *PC-Elektronik Labor, Praxisnahes Lernen mit dem PC als Simulationssystem*, 6. überarbeitete Nachauflage. Poing: RFranzis Verlag GmbH, 2009.

[10] Vitali Mostowoj, *Programmierung eines Virtuellen Oszilloskops mit HTML5 Canvas, Studienarbeit*. Mannheim, 2023.

[11] Peter Fischer, Peter Hofer, *Lexikon der Informatik*, 15. überarbeitete Auflage. Berlin: Springer-Verlag GmbH, 2011.

[12] Ritwick Dey. „Live Server“. (2017), Adresse: https://marketplace.visualstudio.com/items?itemName=ritwickdey.LiveServer (besucht am 12. 03. 2023).

[13] Paul Falstad, Iain Sharp. „Circuit Simulator version 2.7.6js“. (o.J.), Adresse: https://www.falstad.com/circuit/ (besucht am 09. 03. 2023).