

THEMA STUDIENARBEIT

**Javascript: Erweiterung eines Logikeditors für die
Digitaltechnikvorlesung**

im Studiengang

Elektrotechnik

an der Dualen Hochschule Baden-Württemberg Mannheim

von

| | |
|----------------------|--------------------------------|
| Name, Vorname | Epperlein, Justus |
| Abgabedatum | 06.01.2023 |
| Bearbeitungszeitraum | 04.10.2022-22.12.2022 |
| 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: Javascript: Erweiterung eines Logikeditors für die Digitaltechnikvorlesung 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

Inhaltsverzeichnis

Erklärung zur Eigenleistung

Abstract

Inhaltsverzeichnis I

Abkürzungsverzeichnis II

Abbildungsverzeichnis III

Quellcodeverzeichnis IV

Vorwort V

1 Projektbeschreibung 1

2 Vorbetrachtungen 2

2.1 Aktueller Stand der Anwendung 2

2.2 Aufbau des Programms 4

2.3 Vorgehensweise 5

3 Durchführung 6

3.1 Implementierung von XOR und XNOR Logikgattern 6

3.2 Umsetzung der Auswahlfunktion 12

3.3 Integration der Umbenennung von Ein- und Ausgängen 16

3.4 Anpassung der Verbindungsdarstellung 17

3.4.1 Positionierung im Raster 17

3.4.2 Transformation der Verbindungserzeugung 19

3.4.3 Implementierung von Verbindungsabzweigungen 20

3.4.4 Korrektur des Speichervorgangs 21

3.5 Weitere Schritte 21

4 Fazit 22

4.1 Zusammenfassung 22

4.2 Ausblick 22

Literaturverzeichnis 23

Abkürzungsverzeichnis

CSS Cascading Style Sheets

HTML Hypertext Markup Language

D3 Data Driven Documents

DOM Document Object Model

XOR Exklusiv-Oder

XNOR Exklusiv-Nicht-Oder

SVG Scalable Vector Graphics

Abbildungsverzeichnis

| | | |
|---|---|----|
| 1 | Anwendungsoberfläche mit Logikbeispiel | 2 |
| 2 | Modifiziertes Logikbeispiel und Bearbeitungsmenü | 3 |
| 3 | Übersicht über Dateien des Programms | 4 |
| 4 | Struktur der JavaScript Datei script [2, S. 42] | 6 |
| 5 | Wahrheitstabelle XOR-Logikgatter mit drei Eingängen [Vgl. 3, S. 64] . . | 8 |
| 6 | Umwandlung von Symbolen und Begriffen im Parser [Vgl. 1, S. 104 ff.] . | 10 |
| 7 | Auswahlfunktion mit Rahmen und Markierung der Logikelemente | 15 |
| 8 | Änderungen in der Darstellung von Funktionsblöcken | 18 |

Quellcodeverzeichnis

| | | |
|----|---|----|
| 1 | Deklaration von XOR Funktionsblöcken | 7 |
| 2 | Anpassung der replace-Funktion im Bearbeitungsmenü | 9 |
| 3 | Neuer Inhalt des Drop-Down-Menüs in HTML | 10 |
| 4 | Auszug aus Funktion changeOperator | 11 |
| 5 | Definition des HTML-Elements selection [Vgl. 13] | 12 |
| 6 | Bestimmung des Layouts von select [Vgl. 13] | 12 |
| 7 | Deklaration des Elements select in JavaScript [Vgl. 13] | 13 |
| 8 | Mausereignisse und Reaktion auf diese [Vgl. 13] | 14 |
| 9 | Skalierung der Koordinaten entsprechend des aktuellen Zooms | 14 |
| 10 | Anpassen der Eigenschaften des Elements zur Markierung | 15 |
| 11 | Implementierung der rename Option in das Kontextmenü | 16 |
| 12 | Methode renameModule zur Umbenennung | 17 |
| 13 | Kalkulation des nächsten Rasterpunktes | 18 |
| 14 | Zeichnen des Funktionsblocks am nächsten Rasterpunkt | 19 |
| 15 | Erzeugung neuer Abzweigungen durch Ziehen an Leitungen | 21 |

Vorwort

1 Projektbeschreibung

Die Schaffung kombinatorischer Schaltungen durch die Verknüpfung von Logikgattern zur Darstellung von boolescher Algebra ist ein elementarer Bestandteil der Digitaltechnik [Vgl. 1, S. 125]. Diese Studienarbeit beschreibt einen Teilabschnitt der Entwicklung eines Logikeditors zur Erstellung und Simulation von einfachen Logiken für die Digitaltechnikvorlesung. Der Logikanalysator wird als Frontend Webanwendung umgesetzt. Professor Heintz nutzte bislang für die interaktive Einbindung von Logikbaugruppen in seinen Foliensatz den Schaltungssimulator CircuitJS. Da dieser aber als überdimensioniert und schlecht bedienbar empfunden wird und bestimmte Komponenten nicht umfasst, ist es das Ziel dieses Projektes, dass diese Folienelemente in Zukunft in der bearbeiteten Anwendung erstellbar sind. Studenten können zusätzlich Übungen mithilfe des benutzerfreundlichen Logikeditors durchführen. Auf diese Weise kann der Logikeditor den Lernerfolg künftiger Studenten steigern. Angesichts der Corona Pandemie bietet sich eine derartige Anwendung natürlich auch gut für Onlinevorlesungen an.

Diese Arbeit baut auf den Ergebnissen der abgeschlossenen Studienarbeiten von Konstantin Fuchs und Wiebke Albers auf [Vgl. 2, S. 1], welche das Grundgerüst der Anwendung, sowie einige fundamentale Funktionen des Logikanalysators bereits umgesetzt haben. Im Zuge dieser Arbeit soll die Anwendung um mehrere Funktionen erweitert werden, um die Nutzung in der Digitaltechnikvorlesung weiter vorzubereiten. Dazu zählt die Implementierung von Exklusiv-Oder (XOR) und Exklusiv-Nicht-Oder (XNOR) Funktionsblöcken, die Überarbeitung der Verbindungen zwischen Logikelementen inklusive der Ergänzung von Leitungsabzweigungen, die Umsetzung einer Auswahlfunktion zur Markierung und Bearbeitung von mehreren Logikelementen und die Integration der Option zur Umbenennung von Ein- und Ausgangsvariablen.

2 Vorbetrachtungen

Im Folgenden wird der aktuelle Stand der Anwendung in Form der Darstellung des Logikeditors und einiger, für das Projekt relevanter Funktionen beleuchtet. Zudem wird der dafür nötige Aufbau des Programms erläutert.

2.1 Aktueller Stand der Anwendung

In der vorliegenden Version 2.0.6 des Logikeditors lassen sich bereits einige Logikgatter einfügen, miteinander verbinden und die entstandenen Schaltungen simulieren. Dazu besitzt die Anwendung am oberen Rand des Fensters eine Navigationsleiste, die eine Eingabezeile und die Schaltflächen Draw, Logic Elements, Save As, Delete, + und - umfasst [Vgl. 2, S. 35]. Die Benutzeroberfläche der Anwendung mit einer Beispiellojik und einigen geöffneten Menüs ist in Abbildung 1 zu sehen. Mit dem Knopf Draw wird die Eingabe im Arbeitsbereich, der unter der Navigationsleiste liegt, abgebildet, während sich bei Logic Elements ein Drop-down-Menü öffnet, welches eine Auswahl an Logikelementen zum separaten Einfügen in den Arbeitsbereich anbietet. Dazu gehören die Logikgatter AND, NAND, OR, NOR und NOT, sowie Eingänge und Ausgänge bzw. Input und Output. Zusätzlich dazu sollen im Rahmen der Studienarbeit die Gatter XOR und XNOR implementiert werden, um die Auswahl der Basisgatter zu vervollständigen [Vgl. 3, S. 72]. Bei einem Klick auf die Fläche Save As öffnet sich ein weiteres Drop-down-Menü, worüber die gezeichnete Logik als URL, in der Eingabezeile oder als .txt-Datei gespeichert werden kann [Vgl. 2, S. 22 ff.]. Wird der Delete Schalter aktiviert, werden alle Logikelemente, die man anklickt, direkt gelöscht. Mit den Knöpfen + und - lässt sich außerdem der Zoom des Fensters anpassen.

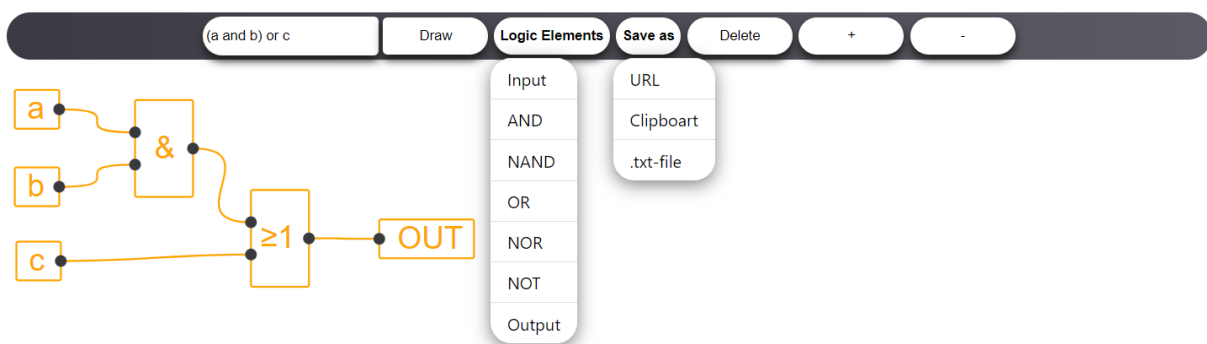


Abbildung 1: Anwendungsoberfläche mit Logikbeispiel

Im Arbeitsbereich befinden sich die gezeichneten Logikelemente, die sich frei per Drag-and-Drop in der Ebene bewegen lassen. Auch die Verbindungen der Elemente lassen sich per Drag-and-drop von den Ausgängen zu den Eingängen ziehen und bilden abhängig vom Anfangs- und Endpunkt eine Kurve, die oft auch unerwünscht andere Elemente im Arbeitsbereich schneidet. **Stattdessen ist vorgegeben, dass** sich die Funktionselemente in Zukunft nur noch auf den Punkten eines Rasters platzieren lassen, damit den Nutzern die Anordnung der Logikelemente erleichtert wird. Die Verbindungen sind zudem eckig darzustellen, um mit existierenden Schaltungssimulatoren, insbesondere dem Logiksimulator Digital, übereinzustimmen. **Außerdem ist vorgesehen,** dass Leitungsabzweigungen eingeführt werden, damit die Darstellung auch bei der Verbindung von einem Ausgang mit mehreren Eingängen möglichst übersichtlich bleibt.

Mit einem Linksklick auf die Inputs können diese aktiviert werden, sodass sie orange erscheinen, oder deaktiviert werden und damit grau gefärbt sind. Basierend auf den Zuständen der Eingänge färben sich auch die Logikblöcke und die Outputs entsprechend. Durch einen Rechtsklick auf ein Funktionselement lässt sich zusätzlich ein Menü öffnen, welches das Löschen des jeweiligen Elements, die Anpassung der Anzahl der Eingänge des Logikblocks oder das Ersetzen mit einem anderen Elemententyp ermöglicht [Vgl. 2, S. 20]. Abbildung 2 zeigt das gleiche Logikbeispiel von oben, aber mit zwei deaktivierten Inputs und dem Bearbeitungsmenü der Logikelemente. Neben den genannten Optionen soll in diesem Menü zudem die Umbenennung von Inputs und Outputs implementiert werden. Wird mit der rechten Maustaste auf den Arbeitsbereich geklickt, öffnet sich ein weiteres Menü, über das die Navigationsleiste ein- und ausgeblendet, der Zoom verändert und, unter dem Punkt About, ein Fenster über die Autoren der Applikation eingeblendet werden kann [Vgl. 2, S. 38].

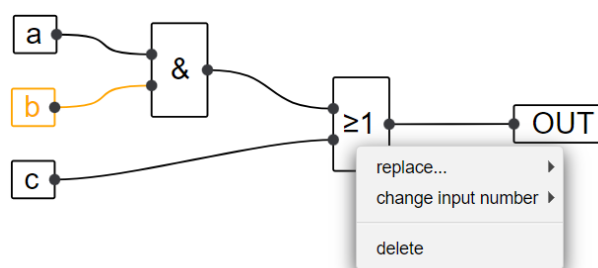


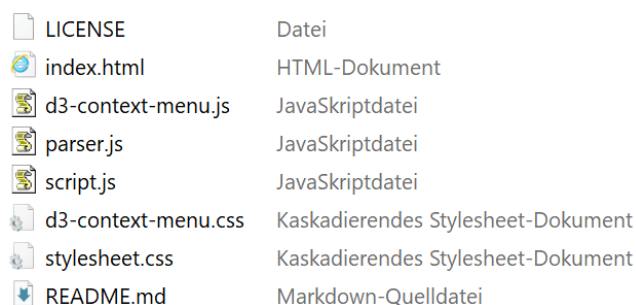
Abbildung 2: Modifiziertes Logikbeispiel und Bearbeitungsmenü

Die Bedienung der Anwendung ist zum aktuellen Stand noch recht aufwendig und nicht benutzerfreundlich, besonders da die typische Editorfunktion zum gleichzeitigen Fangen von mehreren Elementen mithilfe eines rechteckigen Auswahlkastens fehlt. **Diese Funktion ist ebenfalls mit diesem Projekt umzusetzen.** Die interaktive Einbindung

in andere Web-Anwendungen wurde bereits in der vorangegangenen Studienarbeit überprüft [Vgl. 2, S. 40 f.].

2.2 Aufbau des Programms

Wie in Abbildung 3 zu sehen, setzt sich das Programm aus Cascading Style Sheets (CSS)-, JavaScript- und Hypertext Markup Language (HTML)-Dateien zusammen. Der Programmordner enthält zudem die zwei Textdokumente README und LICENSE, die zur groben Beschreibung des Projektes und zur Angabe der Lizenz- und Nutzungsbestimmungen der Anwendung dienen.



| | |
|---------------------|------------------------------------|
| LICENSE | Datei |
| index.html | HTML-Dokument |
| d3-context-menu.js | JavaScriptdatei |
| parser.js | JavaScriptdatei |
| script.js | JavaScriptdatei |
| d3-context-menu.css | Kaskadierendes Stylesheet-Dokument |
| stylesheet.css | Kaskadierendes Stylesheet-Dokument |
| README.md | Markdown-Quelldatei |

Abbildung 3: Übersicht über Dateien des Programms

„HTML [...] ist eine rein textbasierte Auszeichnungssprache zur strukturierten Darstellung von Texten, Grafiken und Hyperlinks in HTML-Dokumenten. HTML-Dokumente können von jedem Webbrowser dargestellt werden und gelten somit auch als Basis für das Internet.“ [4, S. 46] HTML **besitzt** Marken bzw. Tags zur logischen Strukturierung der einzelnen Bereiche eines HTML-Dokuments und gibt damit den Inhalt einer Website vor [Vgl. 4, S. 46]. Die vorliegende Anwendung nutzt die jüngste Version der HTML-Spezifikation, HTML5 [Vgl. 5, S. 713]. Die Darstellung dieser Inhalte bzw. das Layout der Webseite lässt sich eingeschränkt auch in HTML erstellen, wird aber i.d.R. über die Formatvorlagen in CSS umgesetzt [Vgl. 4, S. 47]. Die Formatierung mit CSS bietet mehr Gestaltungsmöglichkeiten gegenüber HTML, erleichtert die nachträgliche Anpassung der Darstellung und vereinfacht die einheitliche Wiedergabe von Webseiten [Vgl. 6, S. 181]. **Der Hauptvorteil von CSS ist aber, dass sich mehrere Elemente gleichzeitig verändern lassen.** Zur Abbildung von Formen und Grafiken wird in der Anwendung Scalable Vector Graphics (SVG) genutzt [Vgl. 2, S. 5], bei dem sich Grafikobjekte nicht aus Pixeln, sondern aus Vektoren zusammensetzen und sich besonders für interaktive Anwendungen anbietet. Durch die Darstellung als Vektorgrafik bleiben die Logikelemente unabhängig vom Zoom immer scharf [Vgl. 4, S. 856].

JavaScript dient als clientseitige Skriptsprache und steuert das Verhalten der Webseite [Vgl. 4, S. 46 ff.]. „Mit JavaScript können Sie die beschränkten Möglichkeiten der Auszeichnungssprache HTML um Benutzerinteraktionen erweitern, um z.B. Inhalte auszuwerten, dynamisch zu verändern oder zu erzeugen.“ [4, S. 48] Für den Zugriff auf die HTML-Elemente eines Dokuments nutzt JavaScript das Document Object Model (DOM) [Vgl. 4, S. 769]. Im HTML-Dokument wird zudem die Bibliothek Data Driven Documents (D3) eingebunden, die die Verarbeitung und Visualisierung von Daten erleichtert [Vgl. 7, S. 13]. „Die Bibliothek erlaubt Ihnen, beliebige Daten mit dem DOM zu verknüpfen und danach datengesteuerte Änderungen auf dieses Dokument anzuwenden.“ [7, S. 13] Zusätzlich ist das D3 Plug-in d3-context-menu integriert, welches für das individuelle Bearbeitungsmenü verantwortlich ist, das sich beim Rechtsklick auf ein Logikelement öffnet, und eine JavaScript- und eine CSS-Datei umfasst. Die beiden JavaScript-Dateien script und parser werden objektorientiert programmiert, d.h. die Struktur des Programms baut auf dem Zusammenspiel von Objekten auf, die Gegenstände repräsentieren und über Attribute bzw. Eigenschaften, sowie Methoden bzw. Verhaltensweisen verfügen [Vgl. 8, S. 30]. Alle genannten Dateien ergänzen sich zusammen zu der Website. Für diese Studienarbeit steht die Bearbeitung des primären JavaScript-Programms script.js im Mittelpunkt, parser.js, index.html und stylesheet.css müssen aber ebenfalls zur Erfüllung der Aufgaben modifiziert werden.

2.3 Vorgehensweise

Mit Visual Studio Code lässt sich der gesamte Programmordner öffnen, wodurch alle Dateien gleichzeitig bearbeitet werden können. Mithilfe der Erweiterung Live Server von Ritwick Dey werden die Dateien als lokaler Webserver ausgeführt [9, Vgl.] und die Anwendung öffnet sich im Webbrowser Microsoft Edge. Dort kann die Bedienung simuliert und die Anpassungen können überprüft werden. Zum Verständnis des vorliegenden Originalcodes und für die Suche nach gewissen Webfunktionen, hauptsächlich in JavaScript, wird das kollaborative, Open-Source-Projekt MDN Web Docs [10] als Nachschlagewerk genutzt, das eine übersichtliche Dokumentation der Technologien von Webplattformen bietet. Das Projekt, welches von der nichtkommerziellen Mozilla Foundation getragen und von verschiedenen Organisationen über das Open Web Docs Kollektiv finanziert wird [11, Vgl.], setzt dabei auf eine unabhängige und unparteiische Informationsvermittlung, mit dem Ziel der Bereitstellung einfach zugänglichen Wissens über Webtechnologien und eines daraus resultierenden Gewinns für die ganze Branche [12, Vgl.].

3 Durchführung

Die Durchführung ist nach den in Abschnitt 2.3 aufgezählten Schritten unterteilt. Im Abschnitt 3.5 werden zusätzliche Änderungen am Programm zur Aktualisierung und zur Fehlerbehebung in der Anwendung beschrieben, sowie mögliche zukünftige Verbesserungen gesammelt.

3.1 Implementierung von XOR und XNOR Logikgattern

Diese Aufgabe erfordert Anpassungen in der HTML-Datei und den JavaScript-Dateien script und parser. Der Aufbau von script.js ist in Abbildung 4 zu sehen. Im Kasten auf der rechten Seite sind die Klassendefinitionen aufgezählt, darunter die Basisklasse Module, die eine Verallgemeinerung der abgeleiteten Klassen, welche alle Arten von Logikelementen charakterisieren, darstellt.

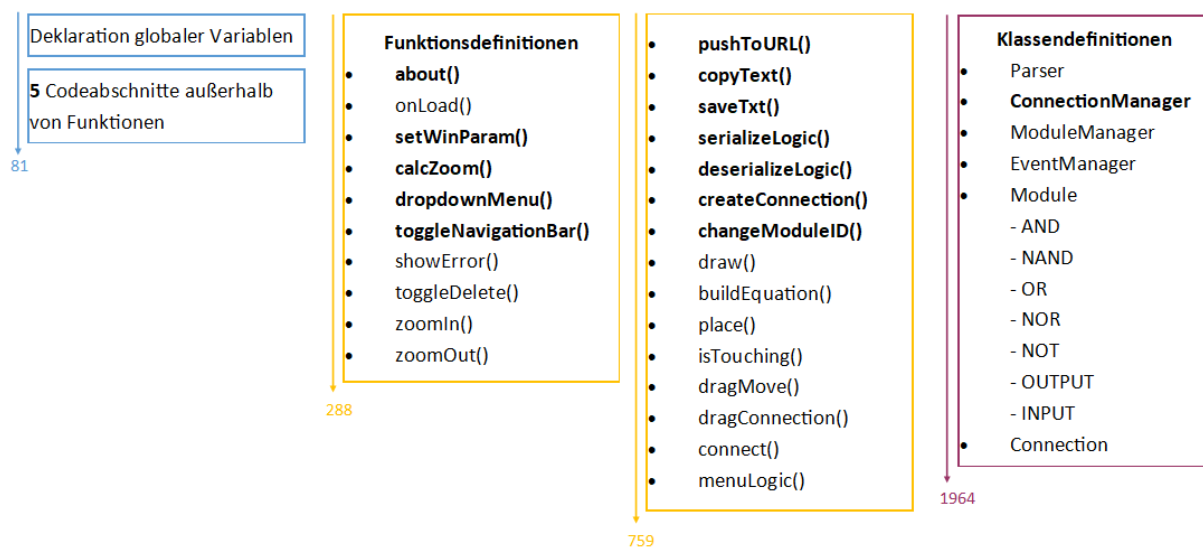


Abbildung 4: Struktur der JavaScript Datei script [2, S. 42]

Die Klassendefinitionen der bisherigen Logikelementarten enthalten jeweils die Darstellungsvorgaben und die Schaltbedingungen der Funktionsblöcke. Die neuen Logikgatter XOR und XNOR müssen demnach auch als Ableitung der Basisklasse Module hinzugefügt werden. Auf Basis der bereits existierenden Schaltgatter werden die abgeleiteten Klassen für XOR und XNOR erstellt. Die Deklaration von XOR ist im Quellcode 1 zu sehen.

```
class XOR extends Module {
  constructor(x, y, inputCount) {
    super(x, y);
    if (!inputCount) {inputCount = 2;}
    this.maxInputCount = inputCount;
    this.maxOutputCount = 1;
    this.width = 30;
    this.height = 50;
    this.calcOutputOffset(false);
    this.calcInputOffset();
    this.build("=1", false);
    this.addsInputToArray();}
  checkActivated() {
    var activate = false;
    for (var i = 0; i < this.input.length; i++) {
      var oneConnectionActive = false;
      for (var j = 0; j < this.input[i].length; j++) {
        if (this.input[i][j].value) {oneConnectionActive = true;}}
      if(oneConnectionActive) {activate = !activate;}}
    this.activateInactivate(activate);}}
```

Quellcode 1: Deklaration von XOR Funktionsblöcken

Die Klassendefinition besteht aus den zwei Methoden constructor und checkActivated. „Bei einem solchen Konstruktor handelt es sich um eine bei der Erstellung einer Objektinstanz aufgerufene Methode, welche die Attributwerte des Objekts in Abhängigkeit der bei ihrem Aufruf übergebenen Parameter initialisiert.“ [8, S. 128] Es werden dementsprechend neuen Logikelementen die Eigenschaften eines XOR-Logikgatters zugewiesen. Dazu werden als Parameter die x- und y-Koordinaten des Elements, sowie die Anzahl der Eingänge übergeben. Mit dem Schlüsselwort super wird der Konstruktor der Basisklasse Module aufgerufen [Vgl. 8, S. 198] und dort die Eigenschaften gegeben, die auf alle Logikelemente zutreffen. Während die Anzahl der Eingänge eines Logikelements von dessen Art abhängt und über das Bearbeitungsmenü zusätzlich angepasst werden kann, besitzen alle bisherigen Elementtypen genau einen Ausgang, was sich aber beispielsweise mit der Einführung von Flipflops ändern könnte. Im Konstruktor der XOR-Funktionsblöcke wird der Variable inputCount deshalb die Anzahl 2 verliehen, falls sie keinen Wert besitzt, wohingegen ihnen die maximale Ausgangszahl 1 zugewiesen wird. Den Logikbausteinen wird zudem die gleiche Breite und Höhe wie den anderen Logikgattern vorgegeben. Mit calcOutputOffset kann die relative Position des Ausgangspunktes zum Funktionsblock berechnet werden, welche davon abhängig ist, ob das Logikgatter über eine Invertierung verfügt. Da in diesem Fall keine Invertierung vorliegt, wird das Argument false übergeben. Mit calcInputOffset wird dagegen die Positionen der Eingänge in Abhängigkeit von deren Anzahl kalkuliert. Über build wird der Funktionsblock gezeichnet, dabei stellt "=1" die Beschriftung

des Elements dar und durch das zweite Argument wird ausgewählt, dass das Logikelement ohne Invertierung gezeichnet wird. Mit `addInputToArray` werden schließlich für die boolschen Werte jedes Eingangs des Funktionsblocks ein Array erstellt, da an den Eingängen wiederum mehrere Verbindungen anliegen können.

Die Methode `checkActivated` bestimmt, welchen Zustand das Logikelement auf Basis der Eingangswerte einnehmen und weitergeben sollte [Vgl. 2, S. 15]. Da die Bedingungen dafür von der Art des Logikgatters abhängig sind, besitzt jede der abgeleiteten Klassen für Funktionsbausteine eine abweichende Methode `checkActivated`. Dabei müssen gewissermaßen sowohl die Ein- als auch die Ausschaltbedingungen definiert sein. Für das vorliegende XOR-Logikgatter werden dazu zunächst die Schaltbedingungen, besonders auch für eine größere Anzahl an Eingängen, betrachtet. In Abbildung 5 ist dafür die Wahrheitstabelle von XOR-Gattern mit den drei Eingängen A, B und C, sowie dem Ausgang D zu sehen.

| A | B | C | D |
|-------|-------|-------|-------|
| false | false | false | false |
| false | false | true | true |
| false | true | false | true |
| false | true | true | false |
| true | false | false | true |
| true | false | true | false |
| true | true | false | false |
| true | true | true | true |

Abbildung 5: Wahrheitstabelle XOR-Logikgatter mit drei Eingängen [Vgl. 3, S. 64]

Anhand der Wahrheitstabelle lässt sich erkennen, dass der Ausgang prinzipiell immer dann einen positiven Zustand annimmt, wenn eine ungerade Anzahl an Eingängen positiv sind. Die Variable `activate` wird also mit dem Wert `false` erzeugt und immer invertiert, wenn ein Eingang positiv ist. Liegen an einem Eingang mehrere Verbindungen an, besteht zwischen ihnen gewissermaßen eine Oder-Verknüpfung, äquivalent zu der Umsetzung dieses Umstandes bei den bestehenden Logikgattern. Zur Überprüfung werden deshalb alle anliegenden Leitungen an allen Eingängen des Funktionsblocks mithilfe von `for`-Schleifen durchlaufen. Befindet sich eine dieser Verbindungen im positiven Zustand wird die Variable `oneConnectionActive` auf `true` gesetzt, was zur Invertierung von `activate` führt. Je nachdem, ob `activate` schließlich positiv oder negativ ist, wird mit `activateInactivate` das Logikelement aktiviert oder deaktiviert.

Bei XNOR-Logikgattern handelt es sich lediglich um XOR-Gatter mit einer Invertierung des Ausgangs [Vgl. 3, S. 64 f.]. Deshalb sind für die Deklaration der abgeleiteten

Klasse XNOR kaum Änderungen von der Klasse XOR nötig. Die Darstellungsvorgaben weichen nur geringfügig von XOR ab. Im Konstruktor müssen calcOutputOffset und build mit dem Argument true übergeben werden, damit die Negation des XNOR-Funktionsblock abgebildet wird. Die Methode checkActivated lässt sich einfach durch die Erzeugung der Variable activate mit dem Wert true anpassen. Auf diese Weise hat XNOR im Vergleich zu XOR bei den gleichen Eingangswerte immer den gegensätzlichen Zustand, was sich mit der logischen Negation deckt.

Im Gegensatz zu den restlichen Drop-Down-Menüs werden die Inhalte des Bearbeitungsmenüs für Logikelemente nicht in HTML definiert, sondern in script.js, weil dieses Menü über das D3 Plug-in d3-context-menu ausgeführt wird. In script.js werden die Optionen und deren Kinder, sowie die durchzuführenden Aktionen definiert und an die JavaScript-Datei d3-context-menu übergeben, wo die Ausführung des Menüs umgesetzt wird. Die replace Option beinhaltet bereits alle bisherigen Arten von Logikelementen, zu denen nun XOR und XNOR hinzuzufügen sind. Der Quellcode 2 zeigt den Kern dieser Definition. In der ersten Zeile wird der angezeigte Name der Unteroption vorgegeben. Der darauffolgende Code bestimmt die Funktion, die bei der Auswahl des Feldes ausgeführt wird. Über die Funktion replace wird das aktuelle Logikelement mit einem neuen XOR-Funktionsblock ersetzt, der sich an der gleichen Position befindet und die gleiche Anzahl an Eingängen besitzt. Dem neuen Element wird außerdem die ID des alten übertragen.

```
title: "XOR",
  action: function () {
    element.replace(new XOR(element.x, element.y, element.maxInputCount),
    element.id);},
```

Quellcode 2: Anpassung der replace-Funktion im Bearbeitungsmenü

Für die XNOR-Logikgatter wird der gleiche Abschnitt wie im Quellcode 2 erneut hinzugefügt, aber der Begriff XOR entsprechend ausgetauscht. Dank der Durchführung dieses Schrittes lassen sich Logikelemente in Zukunft auch mit XOR- und XNOR-Funktionsblöcken ersetzen. Das Schaltverhalten der beiden Funktionsblöcke ließ sich auf diese Weise unter verschiedensten Bedingungen erfolgreich überprüfen.

Damit die neuen Funktionsblöcke auch in der Auswahl des Drop-Down-Menüs Logic Elements in der Navigationsleiste aufgelistet werden, sind Anpassungen an der HTML-Datei erforderlich. Für die XOR-Logikgatter wird übereinstimmend mit den anderen Logikelementen die Zeile in Quellcode 3 erstellt, welche auch für die XNOR-Funktionsbausteine hinzugefügt und entsprechend angepasst wird. Durch sie wird definiert, dass durch einen Klick auf die Option XOR ein neues Modul der Klasse XOR an den Koordinaten (110, 10) platziert wird [10, Vgl.].

```
<a onclick="place(new XOR(110,10))">XOR</a>
```

Quellcode 3: Neuer Inhalt des Drop-Down-Menüs in HTML

Nun steht noch die Anpassung der Eingabeverarbeitung aus, damit die neuen Funktionsblöcke XOR und XNOR auch über die Eingabezeile gezeichnet werden können. Diese Funktion setzt sich aus zwei Segmenten zusammen. Der Parser, der separat in der Datei parser.js umgesetzt ist, erkennt die gewünschten Elemente und wandelt die passenden Eingabeausschnitte in einheitliche Tokens um, damit so verschiedene Eingaben für das gleiche Logikelement möglich sind, diese aber dennoch einheitlich verarbeitet werden können. Die Funktion buildEquation in der JavaScript-Datei script erhält diese vereinheitlichte Zeichenfolge und zeichnet die identifizierte Logikgruppe in den Arbeitsbereich.

Bislang können nur die Logikgatter AND und OR über die Eingabezeile gezeichnet werden [Vgl. 2, S. 44] und die im Parser dafür genutzten Tokens sind nicht eindeutig nachvollziehbar, weshalb vorgesehen ist, dass diese ausgetauscht werden. Der Parser muss also um die Funktionsblöcke NAND, NOR, XOR und XNOR erweitert werden. Als Tokens werden die mathematisch korrekten Symbole der Schaltalgebra für die Logikgatter ermittelt. Die Logikgatter, die möglichen Eingaben zur Auswahl dieser und deren neue Tokens sind in Abbildung 6 zu sehen.

| Logikgatter | Mögliche Eingaben | Parser Token |
|-------------|------------------------------------|--------------|
| AND | *, &, AND, And, and | \wedge |
| NAND | , NAND, Nand, nand | \uparrow |
| OR | +, OR, Or, or | \vee |
| NOR | \, -, NOR, Nor, nor | \downarrow |
| NOT | ~, !, NOT, Not, not | \neg |
| XOR | XOR, Xor, xor | \oplus |
| XNOR | XAND, Xand, xand, XNOR, Xnor, xnor | \odot |

Abbildung 6: Umwandlung von Symbolen und Begriffen im Parser [Vgl. 1, S. 104 ff.]

Auch die Programmierung der Funktion `changeOperator` des Parsers, die für die Umwandlung der Eingabe verantwortlich ist, ist zum aktuellen Zeitpunkt nicht ideal und wird überarbeitet. Die zu suchenden Zeichen und Begriffe, die in der zweiten Spalte von Abbildung 6 benannt sind, bilden das Array `search`, während die Tokens, welche diese ersetzen, an den gleichen Positionen im Array `replace` definiert werden. Dabei wird vor und nach allen zu suchenden Begriffen und deren korrespondierenden Tokens ein Leerzeichen eingefügt, damit es zu keinen Überschneidungen in der Erkennung von Logikgattern oder ähnlich benannten Variablen kommt, zum Beispiel durch `or` statt `nor` usw. Da die möglichen Zeichen die Logikgatter eindeutig vorgeben, ist dies für sie nicht nötig. Wie im Quellcode 4 zu sehen, werden mithilfe einer `for`-Schleife alle Elemente des Arrays `search` durchlaufen und im String `res` mit der `replace` Methode alle Vorkommen des aktuellen `search` Elements mit dem vom Array `replace` ersetzt [10, Vgl.].

```
for(let i = 0; i < search.length;i++) {  
    res = res.replace(search[i], replace[i]);}
```

Quellcode 4: Auszug aus Funktion `changeOperator`

Die restlichen Funktionalitäten in `parser.js` mussten entsprechend an die neuen Tokens angepasst werden, darunter auch die Funktion `checkInputOnEquals`, welches unter anderem für die Erkennung von XNOR-Tokens genutzt wurde, da für diese bislang die Zeichen `==` verwendet wurden. Durch die neue Tokenvergabe konnte diese Funktion vereinfacht werden. Die Funktion `changeSpaces` dient dagegen zum Einfügen eines zusätzlichen AND-Gatters zwischen zwei Variablen, wenn dort sonst kein Operator vorhanden ist. Dies führte allerdings zu einem Fehler, dass auch bei der Eingabe von zwei Klammern hintereinander ein AND-Funktionsblock gezeichnet wurde, welcher gehoben werden konnte.

In `script.js` konnte schließlich die Funktion `buildEquation` modifiziert werden, welche die geparte Eingabe erhält und für das Zeichnen der einzelnen Elemente sorgt. Die Funktion wurde um die Identifikation der neu hinzugefügten Logikgatter erweitert, sowie die den verschiedenen Logikgattern zugewiesenen Tokens aktualisiert.

3.2 Umsetzung der Auswahlfunktion

Zur Umsetzung dieser Funktionalität war der Forumseintrag [13] sehr hilfreich, in welchem der Nutzer burtonator genau nach dieser Thematik fragte. Die Antwort des Users bjb568 auf diesen Eintrag wurde am hilfreichsten eingestuft und sein Profil verfügt über gute Bewertungen, weshalb davon auszugehen ist, dass die von ihm vorgeschlagene Programmierung glaubwürdig ist. Der Autor bjb568 rät zu einer Umsetzung in Form eines neuen Elements in HTML, welches in CSS stilisiert wird. In JavaScript wird schließlich das Verhalten dieses Elements definiert [Vgl. 13]. Es wird zudem ein Link zu einem JSFiddle Dokument präsentiert, in welchem die Programmierung der Antwort entsprechend vorgenommen wurde und diese validiert, da wie gewünscht ein Rechteck zur Auswahl mit der Maus aufgespannt werden kann [Vgl. 13]. Dieses Beispiel bietet eine sehr gute Vorlage für die Bearbeitung dieser Funktion, allerdings sind noch einige Anpassungen vorzunehmen, um Struktur und Bezeichnungen etc. an die bisherige Programmierung der Anwendung anzugleichen. Außerdem umfasst der Vorschlag von bjb568 nicht die Erkennung der in der Auswahl enthaltenen Elemente, welche zu diesem Code hinzugefügt werden muss.

In der HTML-Datei wird das Element select nach dem beschriebenen Vorbild mit den div-Marken eingefügt, wie in Quellcode 5 zu sehen ist. Durch die Einordnung innerhalb des div-Bereichs von frame wird es in dessen Layoutbereich integriert [Vgl. 4, S. 127].

```
<div id="select" hidden></div>
```

Quellcode 5: Definition des HTML-Elements selection [Vgl. 13]

Daneben wird das Layout des Elements select in stylesheet.css angegeben, was in Quellcode 6 abgebildet ist. Der Rahmen wird einen Pixel breit, gepunktet und schwarz dargestellt. „Bei der Verwendung der absoluten Positionierung wird das Element aus dem gewöhnlichen Dokumentenfluss herausgezogen.“ [4, S. 457]

```
#select {  
  border: 1px dotted #000;  
  position: absolute;}
```

Quellcode 6: Bestimmung des Layouts von select [Vgl. 13]

Um der objektorientierten Programmierung der JavaScript-Datei script zu entsprechen, wird von der Vorlage abgewichen und die Variable selectionArea als Objektliteral angelegt [Vgl. 5, S. 125], weil dieses im Gegensatz zu den bisherigen Objekten nur einmal

benötigt wird und somit die Einführung einer neuen Klasse nicht erforderlich ist. Dem Objekt werden die Eigenschaften x1, y1, x2 und y2 für die Koordinaten zugewiesen, die das Rechteck später definieren. Außerdem verfügt das Objekt über die Methoden drawSelection und setSelection. Letztere setzt lediglich alle Koordinaten des Objektes auf neue Werte. Auf die Methode drawSelection wird später ausführlich eingegangen.

Der Auswahlkasten muss beim Drücken der Maus auf den Arbeitsbereich an der aktuellen Position erscheinen und danach seine Größe entsprechend der Bewegung der Maus anpassen, solange diese gedrückt bleibt. Wird die Maus losgelassen, muss der Kasten wieder verschwinden. Die Interaktionen der Maus mit dem HTML-Dokument lassen sich über verschiedene Mausereignisse erfassen und es können Funktionen deklariert werden, die in diesem Fall ausgeführt werden sollen. Wie in Quellcode 8 zu sehen, werden mehrere Typen von Mausereignissen genutzt, um die verschiedenen Phasen des Auswahlverfahrens zu unterstützen. Das Ereignis onmousedown erfolgt beim Drücken der Maus und onmousemove beim Bewegen dieser. Dagegen wird onmouseup im Falle des Loslassens der Maus ausgeführt [Vgl. 5, S. 486].

Wird also eine Maustaste gedrückt und damit onmousedown aktiviert, ist zuerst zu überprüfen, ob sich die Maus unter der Navigationsleiste befindet und keines der Drop-Down-Menüs das Ziel des Mausklicks ist. Mit e.button === 0 kann zudem sichergestellt werden, dass die linke Maustaste gedrückt wurde, damit sich die Funktion nicht beim Rechtsklick auf den Arbeitsbereich einstellt. Sind diese Bedingungen erfüllt, kann das HTML-Element mit select.hidden = 0 sichtbar gemacht werden. Dabei ist hidden der Parameter für die CSS-Eigenschaft visibility [Vgl. 5, S. 459]. Zum Zugriff auf select ist die Deklaration, die in Quellcode 7 zu sehen ist, in script.js erforderlich. Die Koordinaten der Auswahlfläche werden danach auf die des Startpunktes gesetzt. Hier musste jeweils ein bestimmter Offset vorgenommen werden, weil der Auswahlkasten sonst von der tatsächlichen Position der Maus abweicht. Zuletzt wird die Methode drawSelection von selectionArea ausgeführt, die den Auswahlkasten in Abhängigkeit von der Position der Maus zeichnet und alle Logikelemente markiert, die sich im ausgewählten Bereich befinden.

```
var select = document.getElementById("select");
```

Quellcode 7: Deklaration des Elements select in JavaScript [Vgl. 13]

Wenn onmousemove aktiv ist, muss select sichtbar sein, bevor die Koordinaten des Auswahlkastens aktualisiert werden können. Die Variable x2 von selectionArea wird dann immer neu ermittelt, während y2 nur geändert wird, solange sich die Maus unter der Navigationsleiste befindet. Ansonsten erhält es einen Wert, der den Auswahlkasten

unabhängig von der Aktualisierungsdauer der Auswahl immer auf der gleichen Höhe unter der Navigationsleiste enden lässt. Auch hier wird schließlich die Methode drawSelection aufgerufen.

Wird die Maus letztendlich losgelassen, dann wird das select Element wieder versteckt und allen Koordinaten der Wert 0 gegeben. Der Quellcode 8 zeigt die Programmierung zu den beschriebenen Mausereignissen.

```
onmousedown = function(e) {
    if(e.clientY > (widthNavBar + 10) && !e.target.matches('.dropdown-content
a') && !e.target.matches('.d3-context-menu ul li') && e.button === 0) {
        select.hidden = 0;
        selectionArea.setSelection(e.clientX - 10, e.clientY - 65,
e.clientX - 10, e.clientY - 65);
        selectionArea.drawSelection();});
onmousemove = function(e) {
    if(select.hidden == 0) {
        selectionArea.x2 = e.clientX - 10;
        if(e.clientY > (widthNavBar + 10)) {selectionArea.y2 = e.clientY - 65;
        } else {selectionArea.y2 = widthNavBar - 50;}
        selectionArea.drawSelection();});
onmouseup = function(e) {
    select.hidden = 1;
    selectionArea.setSelection(0, 0, 0, 0);};
```

Quellcode 8: Mausereignisse und Reaktion auf diese [Vgl. 13]

In der Methode drawSelection wird der Startpunkt des Auswahlkastens, die Höhe und die Breite des Auswahlkastens kalkuliert und in Pixel umgewandelt, um das HTML-Element select entsprechend dieser Werte anzupassen. **Parallel werden die Koordinaten relativ zum Zoom skaliert**, was für alle vier Koordinaten äquivalent zum Beispielcode 9 erfolgt.

```
x3 = x3*realZoom/(realZoom + viewZoom);
```

Quellcode 9: Skalierung der Koordinaten entsprechend des aktuellen Zooms

Für die tatsächliche Erkennung und Markierung der aktuell ausgewählten Logikelemente werden alle existierenden Elemente durchlaufen und deren Position mit den Grenzen der Auswahl verglichen. Einerseits muss die Auswahl grafisch umgesetzt werden, andererseits benötigen die Logikelemente eine leicht auslesbare Eigenschaft, um festzustellen, ob sie zur Auswahl gehören oder nicht. Zu diesem Zweck wird im Konstruktor der Klasse Module die Variable selected angelegt, die den Wert true besitzt, wenn das entsprechende Element ausgewählt ist. Befindet sich eines der Elemente in

der Auswahl, dann wird dessen Eigenschaft `selected` auf `true` gesetzt und dessen Darstellung angepasst, sodass die Blöcke mit der Farbe `colorSelected` gefüllt werden, für die hellgrau ausgewählt wurde. Die Veränderung dieser Eigenschaften ist in Quellcode 10 zu sehen. Befindet sich das durchlaufene Element außerhalb des Auswahlkastens, so wird ihm gegensätzlich der Wert `false` für `selected` zugewiesen und seine Füllung auf transparent gestellt.

```
element.group.select("rect").attr("fill", colorSelected);  
element.selected = true;
```

Quellcode 10: Anpassen der Eigenschaften des Elements zur Markierung

Abbildung 7 zeigt die Darstellung des Auswahlkastens und der Markierung der ausgewählten Logikelemente in der Anwendung.

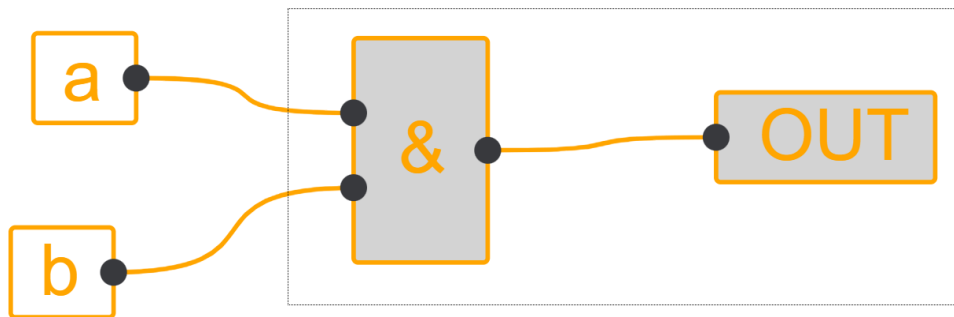


Abbildung 7: Auswahlfunktion mit Rahmen und Markierung der Logikelemente

Die bisherige Umsetzung bietet noch keine Funktionen, die mit der gesamten Auswahl durchgeführt werden können. So ist das gleichzeitige Ziehen und Löschen aller ausgewählten Logikelemente zu ermöglichen. Zur Zusammenführung aller Logikelemente der aktuellen Auswahl in einem Array wird die Funktion `getSelection` erstellt, die als Parameter das angeklickte Modul erhält. Gehört dieses zur Auswahl, durchläuft die Funktion alle existierenden Module und fügt die, bei denen `selected` wahr ist, dem auszugebenden Array hinzu. Wenn das angeklickte Element nicht Teil der Auswahl ist, wird nur dieses eine Logikelement in das Array integriert. Die Funktion `dragMove` zum Ziehen von Logikelementen kann modifiziert werden, dass sie ein Array erhält und die Bewegung für alle enthaltenen Elemente ausführt. Für `delete` ist dies nicht möglich, da es eine Methode von `Module` ist. Stattdessen wird zum Aufrufen der Methode für alle Elemente des zu erhaltenden Arrays die neue Funktion `deleteSelection` ergänzt. Der Aufruf der Funktionen zum Löschen und Ziehen ist den Modifikationen entsprechend anzupassen.

3.3 Integration der Umbenennung von Ein- und Ausgängen

Die Umbenennung von Ein- und Ausgängen, welche neben den Logikgattern ebenfalls abgeleitete Klassen von der Basisklasse Module darstellen, soll als neue Option im Bearbeitungsmenü implementiert werden. Dazu muss die Definition des Kontextmenüs in script.js angepasst werden, wie es bereits in 3.1 geschehen ist. In Quellcode 11 ist die Deklaration der Option rename zu sehen. In der Definition des Kontextmenüs ist es nicht möglich, Bedingungen einzubinden, bzw. in diesem Fall die Optionen des Menüs abhängig von der Art des angeklickten Logikelements zu machen. Dementsprechend wird die Option nicht nur für die Ein- und Ausgänge angezeigt, sondern auch für die Logikgatter, bei denen eine Umbenennung natürlich nicht sinnvoll ist. Wird die Option rename aktiviert, dann wird die neu erstellte Methode von Module, renameModule, ausgeführt. Der Methode werden als Parameter der Name des Konstruktors, der den Typ des Moduls darstellt, sowie die Koordinaten und die ID des Logikelements übergeben.

```
title: "rename",  
action: function () {  
    element.renameModule(element.constructor.name, element.x, element.y,  
        element.id);},
```

Quellcode 11: Implementierung der rename Option in das Kontextmenü

Die Methode renameModule wird in Quellcode 12 gezeigt und gleicht sehr der bestehenden Methode changeInputNumber, da bei beiden ein gewähltes Logikelement mit einem neuen ersetzt wird, welches vom originalen Modul abweichende Eigenschaften besitzt. Bei renameModule wird dabei unterschieden, ob es sich um einen Ein- oder Ausgang oder ein anderes Logikelement handelt. Im Falle eines Ein- oder Ausgangs wird mithilfe der Methode window.prompt() ein Dialogfenster geöffnet, welches die als Parameter übergebene Nachricht anzeigt und ein Eingabefeld umfasst, in das der Anwender den neuen Namen des Ein- oder Ausgangs eingibt [Vgl. 8, S. 234]. Die Variable name entspricht dem ermittelten String. Die eval Funktion dient zur Interpretation von Strings als Quellcode [Vgl. 5, S. 85] und wird hier genutzt, um aus den Variablen einen korrekten Aufruf der replace Methode zu formen [10, Vgl.]. Das ursprüngliche Element wird dabei mit einem neuen Element der gleichen Klasse, an den gleichen Koordinaten und mit dem neuen Namen ersetzt. Dem neuen Element wird zusätzlich die ID des originalen übertragen. Falls es sich bei dem aktuellen Modul aber um ein Logikgatter handelt, wird mit window.alert() eine Nachricht ausgegeben [Vgl. 8, S. 234], die auf die replace Option verweist.


```
renameModule(classname, x, y, oldID) {  
  if(classname == "INPUT" || classname == "OUTPUT") {  
    var name = window.prompt("Rename selected element to...", "");  
    eval("this.replace(new " + classname + "(x, y, name), oldID);");  
  } else {  
    window.alert("Only Inputs and Outputs can be renamed, consider using  
    replace option.");  
  }  
}
```

Quellcode 12: Methode renameModule zur Umbenennung

3.4 Anpassung der Verbindungsdarstellung

Die Überarbeitung der Leitungsdarstellung ist ziemlich komplex und wird deshalb in mehrere Abschnitte unterteilt. Zum einen muss die Positionierung und Darstellung der Logikelemente angepasst werden, um der angestrebten Rasteranordnung zu entsprechen. Zum anderen ist es notwendig, die Verbindungserzeugung zu überarbeiten, um die bisherige Abbildung als Kurven durch eckige Linien zu ersetzen, und das Verhalten dieser Linie zu definieren. Daneben sind Verbindungsabzweigungen einzuführen, die durch das Ziehen an einer bestehenden Leitung zu dem Eingangspunkt eines Logikelements erstellt werden. Schließlich ist die Speicherung der Logik an die vorgenommenen Änderungen zu adaptieren.

3.4.1 Positionierung im Raster

Für das Raster wurde konzipiert, dass sich alle Ein- und Ausgangspunkte der Logikelemente selbst auf Rasterpunkten befinden, wodurch die Linienpunkte wiederum alle auf Punkten des Rasters liegen, was die Positionierung dieser Linienpunkte erleichtert. Als Abstand zwischen zwei Rasterpunkten wurde sich für 15 pt entschieden, weil dies etwa dem bisherigen Abstand der Eingangspunkte bei einem Logikelemente mit zwei Eingängen entspricht. Dieser Wert wurde als globale Variable gridSize angelegt, damit er gegebenenfalls einfach angepasst werden kann. Die Startpunkte, an denen neue Logikelemente erzeugt werden, sowie die Maße der Logikelemente wurden angepasst, dass beide Werte von allen Logikelementen jeweils einem Vielfachen der gridSize entsprechen. Die Maße des Elements können über die Konstruktoren der abgeleiteten Klasse angepasst werden. Gleichzeitig konnte die Höhe von Funktionsblöcken von der Anzahl der Eingänge abhängig gemacht werden. Bei einer hohen Anzahl an Eingängen waren die Eingangspunkte bisher sehr eng beeinander. Im Falle der Invertierung

eines Funktionsblocks wird der Ausgangspunkt weiterhin mit einem gewissen Abstand von der Seite des Blocks gezeichnet, der jeweilige Block ist nun aber schmaler, damit der Ausgangspunkt ebenfalls auf den Rasterkoordinaten liegt. Für diesen Zweck ist es außerdem notwendig, dass die Höhe des Logikgatters immer ein Vielfaches der doppelten gridSize darstellt. Um dies zu gewährleisten, musste die Funktion calcInputOffset modifiziert werden, die die Positionen der Eingangspunkte berechnet, damit sie gleichmäßig über die Höhe des Funktionsblocks verteilt sind. Diese Funktion wird erweitert, dass bei einer geraden Eingangsanzahl kein Eingangspunkt auf der mittleren Höhe gezeichnet wird. Die Abbildung 8 zeigt die Anpassungen in der Darstellung der Funktionsblöcke. In der oberen Reihe ist die bisherige Visualisierung und darunter die korrespondierende neue Darstellung zu sehen.

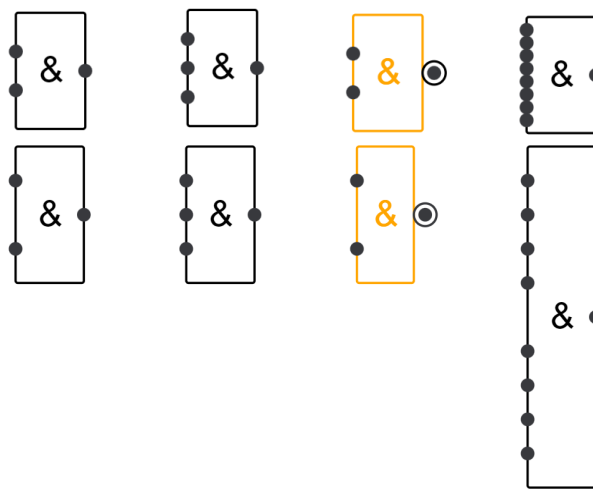


Abbildung 8: Änderungen in der Darstellung von Funktionsblöcken

Zur Bestimmung des nächsten Rasterpunktes wird die Funktion nextGridPoint erstellt, die eine Koordinate coordinate erhält und den Quellcode 13 ausführt, der durch Runden das nächstgelegene Vielfache der gridSize kalkuliert und weitergibt.

```
var res = Math.round(coordinate/gridSize)*gridSize;
```

Quellcode 13: Kalkulation des nächsten Rasterpunktes

Auch während des Bewegens von Logikelementen oder Leitungspunkten dürfen diese sich nur auf Rasterpunkten befinden. Wie in Quellcode 14 zu sehen, wird ein Logikelement, nachdem es durch die Methode dMove bewegt wurde, nicht an dessen tatsächlicher Position gezeichnet, sondern an dem durch die Funktion nextGridPoint kalkulierten nächsten Rasterpunkt. Die Methoden dMoveStart und dMoveEnd für die

Verschiebung der Anfangs- und Endpunkte einer Verbindung werden auf die gleiche Weise angepasst.

```
this.group.attr('transform', 'translate(' + nextGridPoint(this.x) + ', ' +  
nextGridPoint(this.y) + ')');
```

Quellcode 14: Zeichnen des Funktionsblocks am nächsten Rasterpunkt

Werden die Logikelemente nun gezogen, so sind sie zwar immer auf den Rasterpunkten positioniert, ihre wahren Koordinaten, bzw. deren Differenz von den Rasterkoordinaten werden aber zwischen mehreren Bewegungsvorgängen beibehalten, was optisch sehr unangenehm ist. Die wahren Koordinaten müssen darum nach der abgeschlossenen Bewegung auf die Rasterkoordinaten gesetzt werden. Für Verbindung wird aus diesem Grund die Methode latch für das Einrasten in den Rasterpunkten eingeführt, die immer durchgeführt wird, wenn das Ziehen einer neuen Verbindung abgeschlossen ist. Die gleiche Funktion kann für die Logikelemente die Methode dragMove erfüllen, wenn sie einen zweiten Modus erhält, der die Differenz zwischen der dargestellten und der tatsächlichen Position berechnet und das Logikelement um den Wert dieser Differenz bewegt. Dafür wird der Methode zusätzlich eine boolsche Variable übergeben, die den Modus vorgibt.

3.4.2 Transformation der Verbindungserzeugung

Verbindungen wurden bislang über die Koordinaten ihres Start- und Endpunkts definiert. Diese Variablen der Klasse CONNECTION wird durch zwei Arrays für x- und y-Koordinaten der Punkte ersetzt, die die Linie aufspannen, welche die Verbindung darstellt. Die Darstellung der Verbindung erfolgt über das paths Element von SVG und wird verändert, dass sie nicht mehr als Kurve, sondern als Linien zwischen den vier Punkten die sich aus den Werten in den Arrays zusammensetzen, gezeichnet wird. Alle Methoden von CONNECTION, die diese Elemente enthalten, darunter moveEnd, moveStart, dMoveEnd, dMoveStart und latch und der Konstruktor der Klasse werden den Änderungen entsprechend modifiziert. Die Eigenschaft fill von SVG path wurde zudem auf none gestellt, da mit der alten Konfiguration der Innenraum der eckigen Linien einfach transparent ausgefüllt wurde und Interaktionen mit der Leitung auch außerhalb des tatsächlichen Verlaufs der Linie möglich waren. Um trotzdem eine leichte Auswahl der Verbindung zu bieten, wurde die Standardbreite der Linien von einem auf zwei Pixel vergrößert und die verbreiterte Darstellung bei Auswahl von drei auf vier Pixel gesetzt.

Für die neue Leitungsdarstellung wurde ein intelligentes Verhalten ausgelegt, welches in den Methoden `moveEnd`, `moveStart`, `dMoveEnd` und `dMoveStart` der Klasse `CONNECTION` umgesetzt wurde. Obwohl vorerst nur der Anfangs- und Endpunkt der Verbindung durch das Ziehen der verbundenen Logikelemente beweglich ist, ist es dadurch dennoch möglich, den gesamten Verbindungsverlauf eigenhändig festzulegen. Dieses Verhalten basiert auf den folgenden Regeln für die vier Punkte, die eine Leitung definieren:

- Der erste und zweite Punkt liegen immer auf der gleichen Höhe
- Der dritte und vierte Punkt liegen immer auf der gleichen Höhe
- Die y-Koordinaten des zweiten und dritten Punktes sind immer gleich
- Befinden sich Leitungsanfang und -ende auf der gleichen Höhe, wird die horizontale Verschiebung des zweiten und dritten Punktes zurückgesetzt
- Bei einer abgezweigten Leitung sind der erste und zweite Punkt identisch
- Bei einer nicht abgezweigten Leitung hat der zweite Punkt immer mindestens die `gridSize` als horizontalen Abstand zum ersten Punkt
- Beim Löschen der Hauptleitung werden alle abgezweigten Leitungen entfernt
- Eine abgezweigte Leitung bewegt sich mit dem Abschnitt der Hauptleitung mit, an dem sie anliegt

3.4.3 Implementierung von Verbindungsabzweigungen

Die Erzeugung einer neuen Verbindung durch das Ziehen an einer bestehenden Leitung deckt sich funktional grundlegend mit der Erstellung einer Leitung durch das Ziehen an einem Ausgangspunkt eines Logikelements, weshalb sich der dafür verantwortliche Quellcode 15 weitestgehend mit der anderen Stelle im Programm deckt. Zur Erkennung des Ziehens und der damit verbundenen Ereignisse wird die `drag` Funktion von D3 ausgenutzt. Wird der Ziehvorgang begonnen, dann wird eine neue Verbindung erzeugt, deren Koordinaten denen der aktuellen Mausposition `branchStart` entsprechen. Der Eingang der ausgewählten Hauptleitung wird zudem als Eingang der abgezweigten Leitung übernommen. Der Klasse `CONNECTION` wurde die Variable `parent` hinzugefügt, die bei abgezweigten Leitungen die Hauptleitung angibt und an hier definiert wird. Abhängig vom booleschen Wert des Eingangs wird der durch die Verbindung weitergegebene Wert gesetzt. Auf das Ziehen der Leitung wird mit der Funktion `dragConnection` reagiert, welche die Position des Endpunktes aktualisiert. Wird das Ziehen beendet, so wird versucht, die neue Leitung zu verbinden. Wenn dies fehlschlägt, wird

die neue Leitung gelöscht. Andernfalls wird der Ausgangswert der Leitung überprüft. Das Array `branches` wurde ebenfalls als Variable der Klasse `CONNECTION` zugewiesen und umfasst alle von der jeweiligen Leitung abgezweigte Verbindungen. Diese werden über die Methode `addBranch` dem Array `branches` hinzugefügt. Schließlich lässt die Methode `latch` die Verbindung einrasten.

```
.call(d3.drag()  
  .on("start", function () {  
    connection = new CONNECTION([branchStartX, branchStartX, branchStartX,  
    branchStartX], [branchStartY, branchStartY, branchStartY, branchStartY])  
    connection.setInput(thisElement.input);  
    connection.parent = thisElement;  
    if (thisElement.input.value) {  
      thisElement.input.output.push(connection.setValue(true));  
    } else {thisElement.input.output.push(connection.setValue(false));}  
    .on('drag', function () {  
      dragConnection(thisElement.input.output[thisElement.input.output.length -  
      1]);  
    })  
    .on("end", function () {  
      if (!connect(null, thisElement.input.output[thisElement.input.output.length  
      - 1], new EventManager().getLastEventElement(), new  
      EventManager().getElementInputIndex())) {  
        thisElement.input.output.pop().delete();  
      } else {  
        connection.output.checkActivated();  
        thisElement.addBranch(connection);  
        connection.latch();  
      }  
    })  
  });
```

Quellcode 15: Erzeugung neuer Abzweigungen durch Ziehen an Leitungen

Damit alle abgezweigten Leitungen zusammen mit der Hauptleitung gelöscht werden, wurde die Methode `delete` von `CONNECTION` angepasst. Wird eine Leitung gelöscht, wird künftig das gleiche für alle Elemente im Array `branches` durchgeführt. Gleichzeitig wird das Element aus `branches` der Hauptleitung `parent` entfernt, wenn es gelöscht wird.

3.4.4 Korrektur des Speichervorgangs

3.5 Weitere Schritte

4 Fazit

4.1 Zusammenfassung

Die Modifizierung der Eingabezeile durch die Integration von NAND und NOR-Gattern ist für die Vorlesung sehr vorteilhaft, da so einfach nachgestellt werden kann, dass mit jeweils einem dieser Universalgatter alle möglichen Digitalschaltungen erstellt werden können. Diese Erkenntnis ist für die Digitaltechnik sehr bedeutend. FlipFlops siehe dtl S. 140 Nand und Nor dtl S. 110

4.2 Ausblick

Literaturverzeichnis

- [1] Christian Siemers, Axel Sikora, *Taschenbuch Digitaltechnik*, 4. aktualisierte Auflage. München: Carl Hanser Verlag, 2022.
- [2] Wiebke Albers, *Erweiterung eines Logikeditors als Frontend-Anwendung in JavaScript*, Studienarbeit T3100. 2022.
- [3] Vincent Himpe, *Digitale Logik selbst entwickeln, Von 0 und 1 zum FPGA*. Aachen: Elektor-Verlag GmbH, 2012.
- [4] Jürgen Wolf, *HTML und CSS, Das umfassende Handbuch*, 4. aktualisierte und überarbeitete Auflage. Bonn: Rheinwerk Verlag, 2021.
- [5] David Flanagan, *JavaScript, Das umfassende Referenzwerk*, 6. Auflage. Köln: O'Reilly Verlag GmbH & Co. KG, 2012.
- [6] Monika Weber, *HTML + CSS, anfangen, anwenden, verstehen (lernen)*. München: Addison Wesley Verlag, 2003.
- [7] Clemens Gull, *Bigdata mit JavaScript visualisieren, D3.js für die Darstellung großer Datenmengen einsetzen*. Haar: Franzis Verlag GmbH, 2014.
- [8] Jörg Bewersdorff, *Objektorientierte Programmierung mit JavaScript, Direktstart für Einsteiger*, 2. Auflage. Limburg: Springer Fachmedien Wiesbaden GmbH, 2018.
- [9] Ritwick Dey. „Live Server“. (2017), Adresse: <https://marketplace.visualstudio.com/items?itemName=ritwickdey.LiveServer> (besucht am 14. 12. 2022).
- [10] Open Web Docs. „MDN Web Docs“. (o.J.), Adresse: <https://developer.mozilla.org/en-US/> (besucht am 14. 12. 2022).
- [11] Sebastian Grüner. „Google und Microsoft retten MDN-Web-Doku“. (2021), Adresse: <https://www.golem.de/news/mozilla-google-und-microsoft-retten-mdn-web-doku-2101-153668.html> (besucht am 14. 12. 2022).
- [12] Open Collective. „Open Web Docs“. (o.J.), Adresse: <https://opencollective.com/open-web-docs> (besucht am 14. 12. 2022).
- [13] burtonator, bjb568. „Select area/rectangle in javascript“. (2014), Adresse: <https://stackoverflow.com/questions/23284429/select-area-rectangle-in-javascript> (besucht am 10. 12. 2022).