Inhaltsverzeichnis

1 Projektbeschreibung

1.1 Unterrichtsfach

Das Programm kann im Teilbereich „Digitale-Logik“ der Elektrotechnik eingesetzt werden.

1.2 Thema / Projektidee

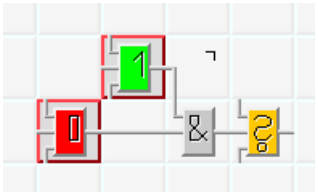
Dieses Programm dient zum Visualisieren, Erzeugen und Testen von digitalen Schaltungen und ist als Teilprogramm (Elektrotechnik-Modul) zum Projekt „ShullBocks“ des BK-Olsberg konzipiert.

1.3 Nutzen für den Unterricht

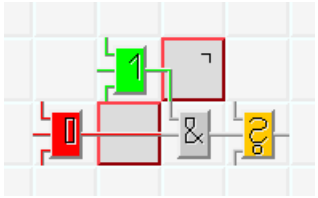
Mit dem Programm läst sich sehr gut die Funktionsweise von einfachen digitalen Schaltungen, aber auch von komplexen Schaltungen mit Rückkopplungen z.B. Flip-Flop zeigen. Zudem kann mit Hilfe von Wertetabellen das Schaltverhalten eigener Schaltungselemente festgelegt werden. Dadurch kann man z.B. eigene Symbole mit dem Verhalten eines J-K-Flip-Flops betreiben. Auf diese Weise können mit dem Programm letztendlich auch Zählwerke etc. generiert und simuliert werden. Da die Schaltungen gespeichert und geladen werden können, kann somit eine Sammlung unterschiedlicher Schaltungen aufgebaut werden. Schaltungen können auch ausgedruckt werden, wobei das Programm selbstständig den Ausdruck auf die gegebene Papiergröße optimiert. Der Ausdruck eignet sich durch diese Optimierung sehr gut als Kopiervorlage oder zum Erstellen von Folien für den Unterricht.

2 Lösungskonzept

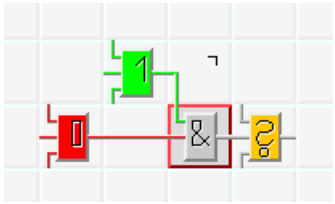
2.1 Aufbau der Lösung



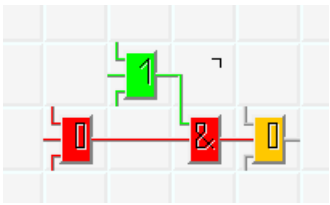
In einem 2 dimensionalem Feld mit 40x40 Zellen (bzw. Rahmen) können beliebige Schaltsymbole und Verbindungselemente eingesetzt werden. Die Berechnung beginnt bei den Startwerten, die je nach logischem Wert (0 = rot; 1 = grün) die entsprechenden Farben angenommen haben.



Wenn anschießend ein Berechnungs-„Schritt“ ausgeführt wird, werden die Startwerte an die benachbarten Rahmen übertragen. Auch diese verändern ihre Farbe gemäß den Logischen Werten (0 = rot; 1 = grün);



Rahmen, die im nächsten Berechnungsschritt berechnet werden, werden mit einem Roten Rand (Farbe kann verändert werden) hervorgehoben. Um die höhere Verzögerungszeit eines Schaltsymbols gegenüber eines Verbindungselementes zu simulieren, geben Schaltsymbole erst nach 10 Schritten (Verzögerungszeiten können verändert werden) den berechneten Wert an den Folgerahmen weiter.



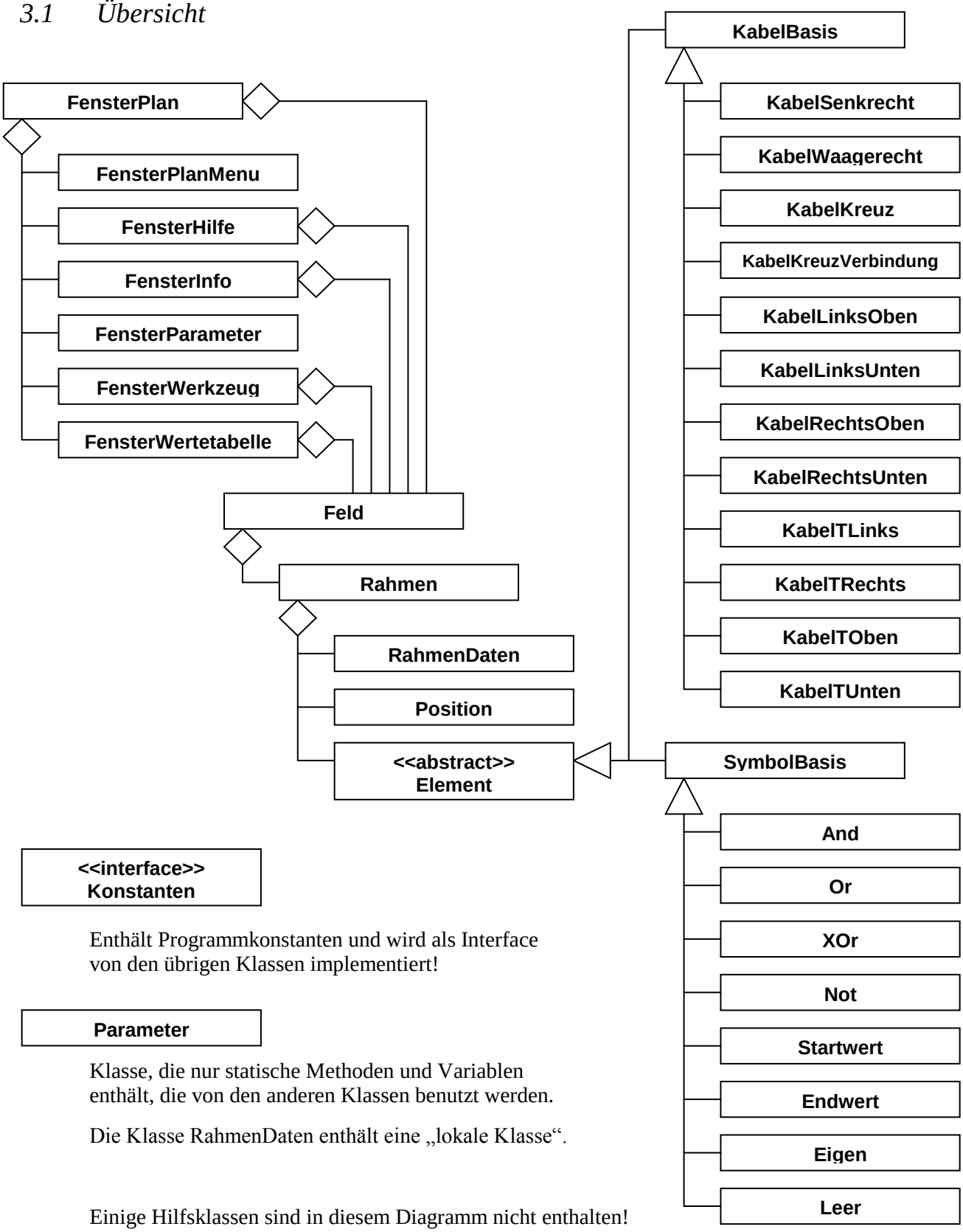
Auf diese Weise kommt erst nach 12 Schritten der Wert 0 bei dem Endrahmen (Zelle mit dem Fragezeichen) an. Der Endrahmen zeigt dann den ankommenden Wert an und die Berechnung ist abgeschlossen.

2.2 Eingesetzte Verfahren

Die Werte werden in der Schaltung iterativ von Rahmen zu Rahmen weitergereicht, da eine rekursive Lösung nicht dem Verhalten tatsächlicher Schaltungen entspricht und bei aufwendigen Schaltungen (mit Rückkopplung oder mit Taktsteuerung) zu falschen Ergebnissen führen würde.

3 Programm-Architektur

3.1 Übersicht



<<interface>> Konstanten

Enthält Programmkonstanten und wird als Interface von den übrigen Klassen implementiert!

Parameter

Klasse, die nur statische Methoden und Variablen enthält, die von den anderen Klassen benutzt werden.

Die Klasse RahmenDaten enthält eine „lokale Klasse“.

Einige Hilfsklassen sind in diesem Diagramm nicht enthalten!

3.2 Funktionsweise

Bereits bei dieser Komplexität des Programms ist es wenig sinnvoll jede Klasse und jede Methode zu beschreiben! In dem Programm wurden Methodennamen verwendet, die in der Regel sehr einfach auf deren Aufgabe und Funktionsweise schließen lassen. Im Folgenden wird daher exemplarisch die zentrale Berechnungsmethode der Klasse „Feld“ genauer erläutert. Bevor jedoch deren Funktionsweise verstanden werden kann, muss der Aufbau der Rahmen und deren Objektvariablen genauer untersucht werden.

3.2.1 Daten eines Rahmens

Jeder Zelle des Schaltplanes (des Feldes) ist ein Rahmen“ zugewiesen. Einem Rahmen wiederum enthält folgende Informationen (Objektvariablen)

```
protected Position    position;      // x und y Koordinaten
protected int         darstellung;   // NORMAL, UNSICHTBAR, HERVORGEHOBEN
protected Element     element;       // Kabel oder Schaltobjekt
protected RahmenDaten rahmenDaten;   // Daten zum Rahmen
```

Dabei sind die Variablen „position“, „element“ und „rahmenDaten“ wiederum Objekte mit eigenen Objektvariablen:

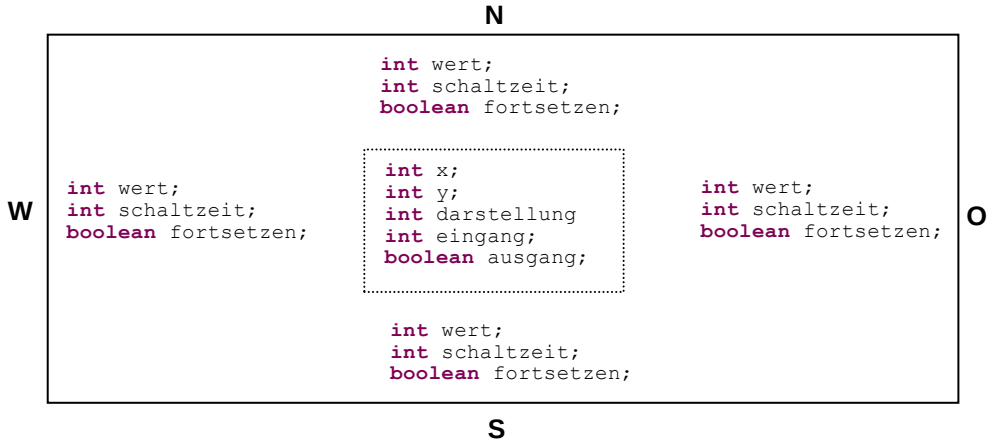
```
protected Position    position;      // x und y Koordinaten
        protected int x;
        protected int y;
protected int         darstellung;   // NORMAL, UNSICHTBAR, HERVORGEHOBEN
protected Element     element;       // Kabel oder Schaltobjekt
        protected int eingang;       // Anzahl der Eingaenge
        protected boolean ausgang;   // false = der Ausgang muss negiert werden
protected RahmenDaten rahmenDaten;   // Daten zum Rahmen
        protected Daten norden;
        protected Daten sueden;
        protected Daten westen;
        protected Daten osten;
```

Hierbei setzt sich ein „RahmenDaten“-Objekt wiederum aus vier Objekten der lokalen Klasse „Daten“ zusammen, die die Objektvariablen „wert“, „schaltzeit“ und „fortsetzen“ besitzen:

```
protected Position    position;      // x und y Koordinaten
        protected int x;
        protected int y;
protected int         darstellung;   // NORMAL, UNSICHTBAR, HERVORGEHOBEN
protected Element     element;       // Kabel oder Schaltobjekt
        protected int eingang;       // Anzahl der Eingaenge
        protected boolean ausgang;   // false = der Ausgang muss negiert werden
protected RahmenDaten rahmenDaten;   // Daten zum Rahmen
        protected Daten norden;
                protected int wert;
                        // UNGUELTIG, UNBEKANNT, UNBEKANNT_AUSGANG, EINS, NULL
                protected int schaltzeit;
                        // UNGUELTIG, UNBEKANNT, NULL (sofort schalten), Schaltzeit
                protected boolean fortsetzen;
                        // true = Fortsetzung in diese Richtung erforderlich
        protected Daten sueden;
                protected int wert;
                protected int schaltzeit;
                protected boolean fortsetzen;
        protected Daten westen;
                protected int wert;
                protected int schaltzeit;
                protected boolean fortsetzen;
        protected Daten osten;
                protected int wert;
                protected int schaltzeit;
                protected boolean fortsetzen;
```

Wichtig ist, dass für alle Richtungen eines Rahmens also für Norden, Süden, Westen und Osten drei unterschiedliche Werte gespeichert werden können. Zum einen kann für jede

Richtung der „Wert“ gespeichert werden. Neben den logischen Werten 0 und 1 sind hier aber auch die Werte „ungültig“, „unbekannt“ und „unbekannt_Ausgang“ möglich (hierbei handelt es sich um Konstanten, deren Werte in dem Interface „Konstanten“ festgelegt worden sind). Zum anderen kann auch für jede Richtung angegeben werden nach welcher Schaltzeit der Wert berechnet, das heißt an die anderen Ausgänge weitergereicht werden muss. Zudem kann für jede Richtung mit dem booleschen Wert „fortsetzen“ angegeben werden, an welche Nachbarrahmen der Wert weitergegeben werden muss. Anschaulich kann man sich die Daten eines Rahmens wie folgt vorstellen:



Dieser Aufbau war unter anderem notwendig, da bei einer Leitungskreuzung ein Rahmen unterschiedliche Werte annehmen kann. Beispielweise kann der Rahmen in einem solchen Fall in der Nord-Süd Richtung einen anderen logischen Wert als in West-Ost-Richtung haben. Ebenso muss ein neuer Wert, der von Norden an eine Leitungskreuzung weitergegeben wird nur nach Süden und nicht an den westlichen und östlichen Ausgang weitergereicht werden.

3.2.2 Zentrale Berechnungsfunktion

```
1  public void berechneFeldSchritt ()
2  {
3     boolean berechnung;
4
5     Parameter.berechnungsSchritt++;
6
7     do
8     {
9        berechnung = false;
10
11       for (int x=0; x<feld.length; x++)
12       {
13          for (int y=0; y<feld[x].length; y++)
14          {
15             if (feld[x][y].testSchaltzeitNull())
16             {
17                berechnung = true;
18                feld[x][y].berechneWert();
19             }
20          }
21       }
22       for (int x=0; x<feld.length; x++)
23       {
24          for (int y=0; y<feld[x].length; y++)
25          {
26             if (feld[x][y].testFortsetzen())
27             {
28                if (feld[x][y].getFortsetzen(NORDEN) && y-1 >= 0)
29                {
30                   if (feld[x][y-1].setWert (SUEDEN, feld[x][y].getWert(NORDEN)))
31                   {
32                      feld[x][y-1].setSchaltzeit(SUEDEN);
33                   }
34                }
```

Kontrolliert, ob Schaltzeiten abgelaufen sind und eine Berechnung erfolgen muss.

Kontrolliert, ob Werte an Nachbarfelder weitergereicht werden müssen.

```
35                if (feld[x][y].getFortsetzen(SUEDEN) && y+1 < feld.length)
36                {
37                   if (feld[x][y+1].setWert (NORDEN, feld[x][y].getWert(SUEDEN)))
38                   {
39                      feld[x][y+1].setSchaltzeit(NORDEN);
40                   }
41                }
42                if (feld[x][y].getFortsetzen(WESTEN) && x-1 >= 0)
43                {
44                   if (feld[x-1][y].setWert (OSTEN, feld[x][y].getWert(WESTEN)))
45                   {
46                      feld[x-1][y].setSchaltzeit(OSTEN);
47                   }
48                }
49                if (feld[x][y].getFortsetzen(OSTEN) && x+1 < feld.length)
50                {
51                   if(feld[x+1][y].setWert (WESTEN, feld[x][y].getWert(OSTEN)))
52                   {
53                      feld[x+1][y].setSchaltzeit(WESTEN);
54                   }
55                }
56                feld[x][y].resetFortsetzen();
57             }
58          }
59       }
60    }
61    while (berechnung);
62
63    for (int x=0; x<feld.length; x++)
64    {
65       for (int y=0; y<feld[x].length; y++)
66       {
67          feld[x][y].berechneSchaltzeit();
68       }
69    }
70 }
```

Die Methode „berechneFeldSchritt()“ führt einen Berechnungsschritt für das gesamte Feld aus. Da auch eine Schaltzeit von „0“ eingestellt werden kann, was bedeutet, dass die Kabelverbindungen ohne Zeitverzögerung den berechneten Wert weiterreichen sollen, muss diese Funktion, solange noch irgendein Schaltelement mit Schaltzeit „0“ existiert, mit der Berechnung fortfahren. Daher müssen in einer Schleife immer wieder alle Schaltzeiten der Felder kontrolliert werden, bis kein Feld mehr eine Schaltzeit von „0“ hat. Die lokale Variable „berechnung“ wurde eingeführt um genau diese Kontrolle zu realisieren. Die Variable nimmt so lange den Wert „true“ an, wie es noch Felder mit Schaltzeit = 0 gibt.

Zeile 3: Die boolean-Variable „berechnung“ wird deklariert.

Zeile 5: Die globale Variable „berechnungsSchritt“ dient zum Zählen der Berechnungsschritte und wird um eins erhöht.

Zeilen 11-21: Schleifenkonstrukt zum Durchlaufen aller Felder.

Zeile 15: Mit der Funktion „testSchaltzeitNull“ wird kontrolliert, ob ein Rahmen eine Schaltzeit von „0“ besitzt. Die Funktion liefert den Wert „true“, sobald bei den Rahmendaten eine der Richtungen Norden, Süden, Westen oder Osten die Schaltzeit = 0 besitzt. Ist dies der Fall, ist für dieses Element die Verzögerungszeit abgelaufen und die Berechnung muss erfolgen.

Zeile 17: Da in diesem Fall eine Berechnung erfolgen muss, wird die Variable „berechnung“ auf „true“ gesetzt.

Zeile 18: Die tatsächliche Berechnung wird ausgeführt. Da die Berechnung natürlich von dem Schaltsymbol abhängt, gibt es auch in jeder Schaltsymbolklasse („And“, „Or“, etc.) eine eigene Funktion „berechneWert“, die folgende Aufgaben ausführt:

1. Berechnet ausgehend von den anliegenden Eingangswerten den Ausgangswert und stellt diesen ein.

2. Legt fest in welche Richtungen der Berechnete Wert an benachbarte Rahmen weitergegeben werden muss.

3. Setzt alle eigenen Schaltzeiten auf den Ausgangszustand zurück.

Wie diese Funktion beispielsweise für ein And-Gatter aussieht ist im Kapitel 6 zu sehen.

Diese Funktionen werden aufgrund des Polymorphismuss passend zugeordnet, da sie als abstrakte Funktion in der abstrakten Klasse „Element“ verankert ist. Das fehlende Bindeglied zwischen dieser Funktion und der Funktion „berechneWert“ der Schaltsymbolklassen ist in der Klasse Rahmen zu finden:

```
public int berechneWert ()
{
        return this.element.berechneWert(this.rahmenDaten);
}
```

Zeile 22-59: Schleifenkonstrukt zum Durchlaufen aller Felder.

Zeile 26: Kontrolliert, ob bei einer der Richtungen Norden, Süden, Westen oder Osten der Wert der Variable „fortsetzen“ auf „true“ gesetzt ist.

Zeile 28: Ist dies der Fall wird kontrolliere, ob der Wert „fortsetzen“ der Richtung Norden auf „true“ gesetzt ist und es ein nördlichen Rahmen gibt.

Zeile 29: Ist dies der Fall wird versuche im Suden des nördlichen Rahmens den nördlichen Wert des eigenen Rahmens einzutragen.

Zeile 32: War dies möglich wird auch die südliche Schaltzeit des nördlichen Rahmens entsprechend eingestellt. Da die Schaltzeiten abhängig sind vom Schaltsymbol gibt es auch hier in jeder Schaltsymbolklasse („And“, „Or“, etc.) eine eigene Funktion „setSchaltzeit“ die analog zur Funktion „berechneWert“ angesprochen wird.

Zeilen 35-53: Analog zu Zeilen 26-32

Zeile 56: Da die Werte jetzt an die Nachbarrahmen weitergereicht worden sind, müssen die Variablen „fortsetzen“ des eigenen Feldes zurückgesetzt werden.

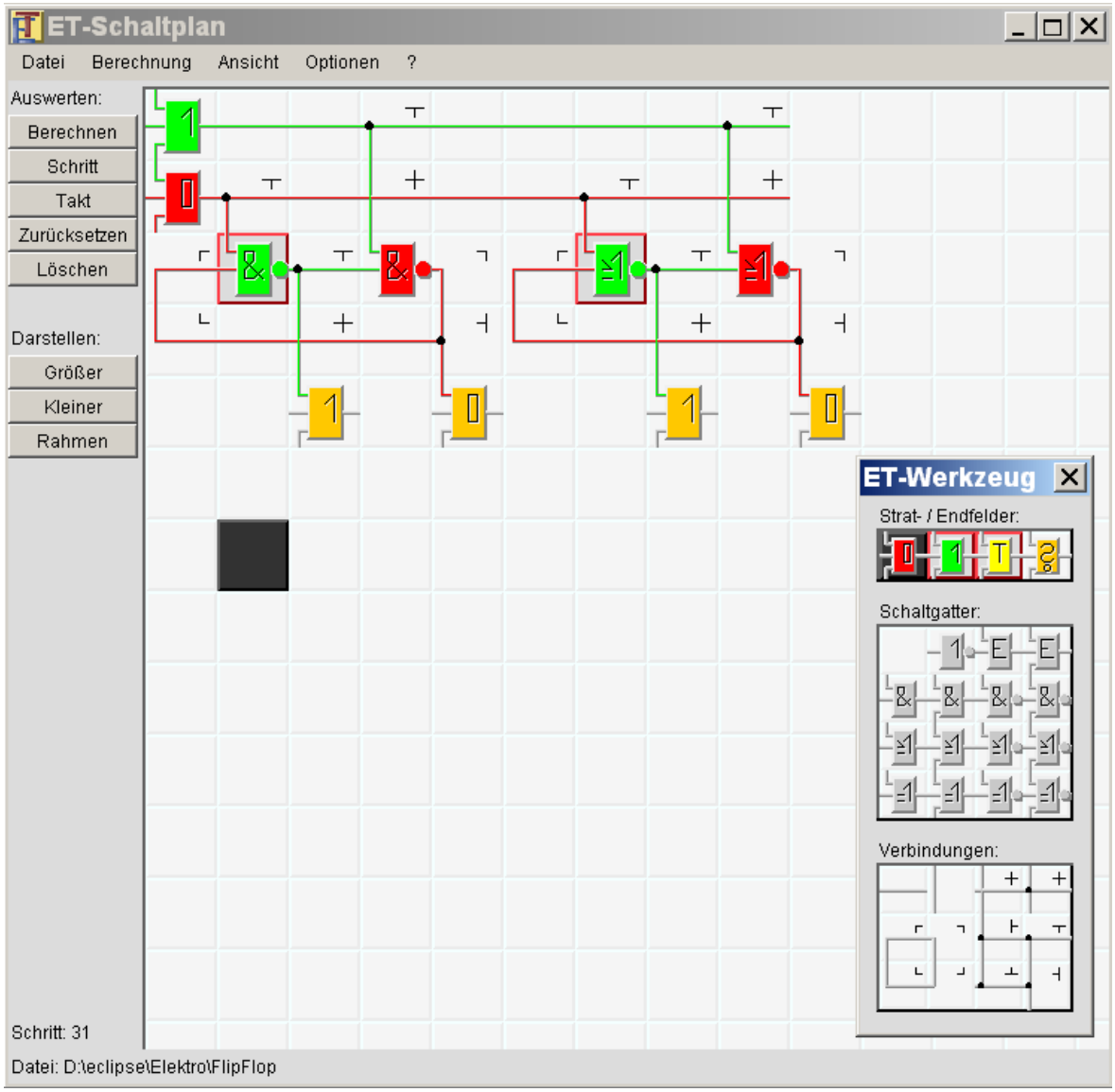
Zeile 61: Diese Programmschritte werden so lange wiederholt, bis keine Schaltzeiten mit Wert 0 mehr existieren, das heißt, dass für diesen Berechnungsschritt keine Berechnungen mehr ausgeführt werden müssen.

Zeile 63-59: Schleifenkonstrukt zum Durchlaufen aller Felder.

Zeile 67: Da nun ein Berechnungsschritt abgeschlossen ist müssen für alle Felder die Schaltzeiten (wenn vorhanden) um eins verringert werden.

4 Benutzerschnittstelle

4.1 Konzept

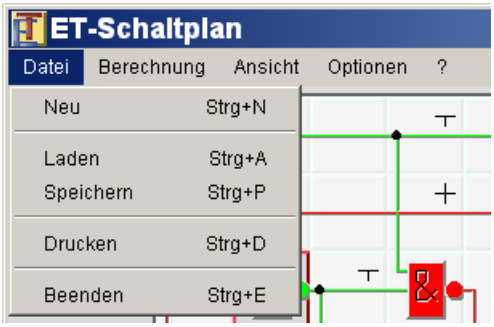


In einem Feld von maximal 40x40 Zellen können elektronische Schaltelemente und Kabelverbindungen positioniert werden. Dabei können alle Objekte des ET-Werkzeugfensters beliebig auf dem Schaltplan gesetzt werden. Auf diese Weise können selbst komplexe Schaltungen wie ein Puzzle zusammengebaut werden. Nach Fertigstellung der Schaltung kann mit Hilfe der Funktionen „Berechnen“ „Schritt“ und „Takt“ der Wert und der Ausgang der Schaltung berechnet werden. Dabei färben sich die Kabelstrecken und auch die Schaltelemente nach und nach in den Farben rot (für den logischen 0-Wert) oder grün (für den logischen 1-Wert). Diese Berechnung funktioniert auch bei Rückkopplung, wobei die Berechnung erst dann abbricht, wenn die Schaltung entweder einen stabilen Wert annimmt oder eine vom Benutzer festlegbare Maximalschrittzahl überschreitet. Auf diese Weise lässt sich mit dem Programm sehr gut die Funktionsweise z.B. von Flip-Flops zeigen. Zudem lassen sich für das Schaltsymbol E (für eigenes Schaltsymbol) über Wertetabellen das Schaltverhalten eigenständig definieren. Dadurch kann man z.B. das E-Symbol mit dem Verhalten eines J-K-Flip-Flops betreiben. Auf diese Weise können mit dem Programm letztendlich auch Zählwerke etc. generiert und simuliert

werden. Da die Schaltungen gespeichert und geladen werden können, kann somit eine Sammlung unterschiedlicher Schaltungen aufgebaut werden. Schaltungen können auch ausgedruckt werden, wobei das Programm selbstständig den Ausdruck auf die gegebene Papiergröße optimiert. Der Ausdruck eignet sich durch diese Optimierung sehr gut als Kopiervorlage oder zum Erstellen von Folien für den Unterricht.

Aber nun zu den Funktionen im Einzelnen:

4.2 *Dateifunktionen*



4.2.1 *Neu*

Mit der Funktion „Neu“ kann mit einer neuen Schaltung begonnen werden.

4.2.2 *Laden / Speichern*

Schaltungen können mit beliebigem Namen gespeichert werden. Dabei werden alle Schaltungselemente aber auch alle Einstellungsinformationen wie Größe, Farbe, Parameter (siehe Optionen) und Darstellung gespeichert.

Beim Laden werden daher neben der Schaltung auch die Einstellungen wieder hergestellt.

Die Funktionen „Laden“ und „Speichern“ benutzen die Systemdialoge.

4.2.3 *Drucken*

Die Funktion „Drucken“ erlaubt zunächst das Einstellen der Papiergröße, Kopienzahl etc. über Systemdialoge.

Anschließend wird die Schaltung auf die gegebene Seitengröße optimal angepasst und ausgedruckt (ohne Feldraster).

4.2.4 *Beenden*

Beendet das Programm.

4.3 Berechnungsfunktionen

4.3.1 Berechnen

Bei einer Berechnung werden standardmäßig bis zu 100 Einschritte (siehe 4.3.2 Schritt) ausgeführt. Diese Anzahl von 100 Schritten je Berechnung kann vom Benutzer verändert werden. Bei einfachen Schaltungen ist in der Regel damit die Berechnung abgeschlossen.

4.3.2 Schritt

Ausgehend von den Startwerten werden die logischen Werte an die benachbarten Rahmen weitergereicht. Diese Rahmen können eine Verzögerungszeit >= 1 haben. Rahmen mit einer Verzögerungszeit von 10 geben den berechneten Wert z.B. erst nach 10 Schritten weiter. Standardmäßig ist bei einem Kabelelement die Verzögerungszeit auf 1 eingestellt, während die Schaltsymbole standardmäßig eine Verzögerungszeit von 10 haben. Diese Werte können jedoch vom Anwender verändert werden. Bei einem Schaltsymbol mit mehreren Eingängen wird die Berechnung bereits angestoßen, wenn an einem der Eingänge der Wert Null oder Eins ankommt (Dies ist beispielsweise für Flip-Flops notwendig). Bei alle anderen Eingänge mit noch unbekanntem Eingang wird für die Berechnung von dem logischen Wert 0 ausgegangen.

4.3.3 Takt

Neben den Startwerten „Null“ und „Eins“ gibt es noch den Startwert „T“ für Takt. Diese Funktion „Takt“ des Menüs ändert den logischen Wert des Takt-Startwertes von Null auf Eins bzw. umgekehrt.

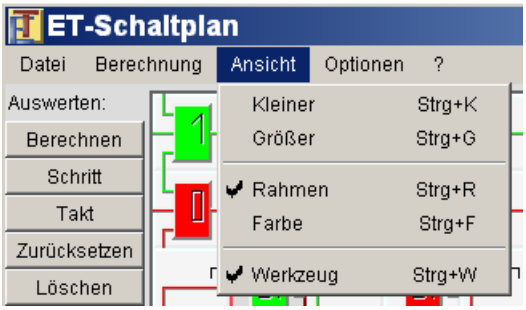
4.3.4 Zurücksetzen

Durch diese Funktion wird die Schaltung auf den Ausgangswert zurückgesetzt, d.h. alle Kabelverbindungen und Schaltsymbole, die berechnet worden sind (und somit die Farben rot oder grün angenommen haben), werden auf einen undefinierten Wert zurückgesetzt und somit in der Schaltung wieder Grau angezeigt.

4.3.5 Löschen

Die gesamte Schaltung wird gelöscht! Möchte man nur einzelne Felder des Schaltplans löschen, muss aus dem Werkzeugfenster das „leere“-Symbol ausgewählt werden, mit dem dann die zu löschenden Elemente des Schaltplans überschrieben werden können.

4.4 Ansichtsfunktionen



Für den Schaltplan steht ein Feld von 40x40 Rahmen zur Verfügung. Um für kleinere Schaltungen nur einen Teilbereich oder einen Ausschnitt einer größeren Schaltung betrachten zu können, kann die Darstellung entsprechen angepasst werden. Der Cursor verbleibt dabei immer im sichtbaren Bereich! Verlässt man mit dem Cursor den sichtbaren Bereich, wird die Schaltung entsprechend gescrollt.

4.4.1 Kleiner

Verkleinert die Darstellung der Schaltungselemente.

4.4.2 Größer

Vergrößert die Darstellung der Schaltungselemente.

4.4.3 Rahmen

Die Rasterung des Feldes und die Anzeige des Cursors können ein- und ausgeblendet werden.

4.4.4 Farbe

Für die Darstellung können vier verschiedene Farbeinstellungen gewählt werden:

Grau, Orange, Grün, Blau

Mit dieser Funktion „Farbe“ kann zwischen den vier Farbeinstellungen gewechselt werden.

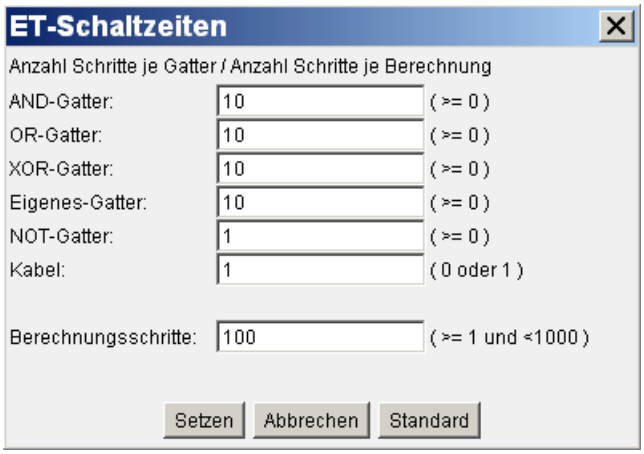
4.4.5 Werkzeug

Das Wekzeug-Fenster kann mit dieser Funktion ein- und ausgeblendet werden.

4.5 *Optionen-Funktionen*

Unter Optionen sind Funktionen zum Einstellen von Parametern zu finden.

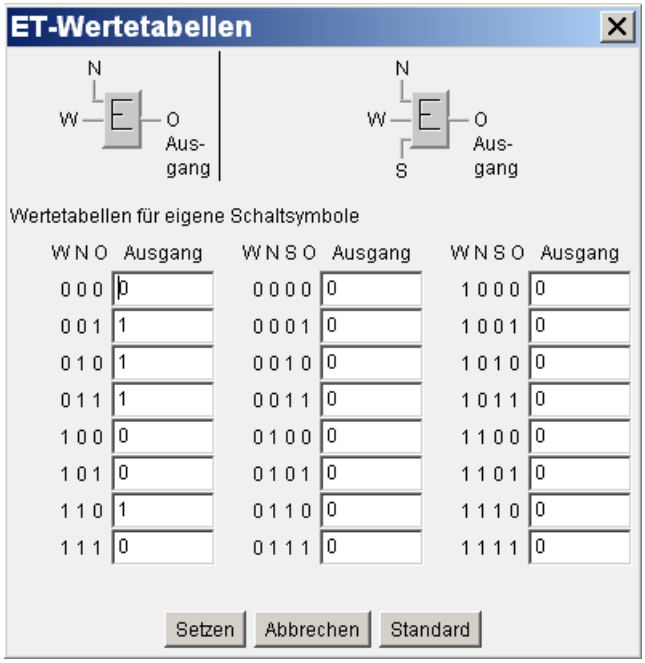
4.5.1 *Schaltzeiten*



Hier lassen sich die Verzögerungs- bzw. Schaltzeiten für die Schaltsymbole und Kabelverbindungen festlegen. Der Wert muss >=0 sein und gibt an, wie viele „Schritte“ ausgeführt werden müssen bevor der Wert an den Folgerahmen weitergegeben wird. Verändern sich während der Verzögerungszeit die Werte an den Eingängen, so werden diese Veränderungen bei der Berechnung des Ausgangswertes berücksichtigt.

In dem Feld „Berechnungsschritte“ kann angegeben werden wie viele Einzelschritte beim Ausführen der Berechnung durchgeführt werden sollen. Diese Einstellung verhindert, dass bei einer alternierenden Schaltung das Programm in eine Endlosberechnung gerät! Der Wert muss zwischen 1 und 1000 liegen.

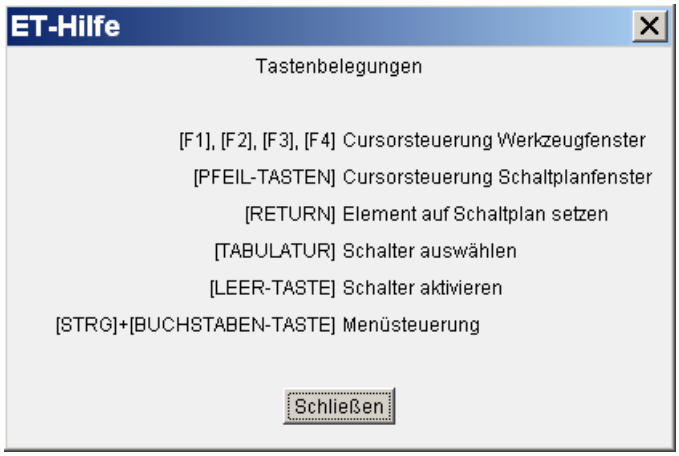
4.5.2 Wertetabellen



Hier kann mit Hilfe von Wertetabellen das Schaltverhalten eigener Schaltsymbole festgelegt werden. Dabei kann auch der Ausgangswert des Schaltsymbols als Eingangswert des Folgewertes benutzt werden! Als Eingabewerte sind lediglich die logischen Werte „0“ und „1“ zulässig! Als Standardwerte sind für ein eigenes Schaltsymbol mit zwei Eingängen und Rückkopplung beispielhaft die Werte eines J-K-Flip-Flops vorgegeben. Die Beschriftung „N“, „S“, „W“, „O“ steht für Norden, Süden, Westen, Osten und gibt somit die Richtung des Eingangs bzw. Ausgangs an.

4.6 *Hilfsfunktionen*

4.6.1 *Hilfe*



Unter dem Menüpunkt „Hilfe“ sind die Tastenbelegungen zu Finden, die zur Steuerung des Programms benutzt werden können. Bemerkenswert ist, dass aus dem Schaltplan Fenster heraus auch der Cursor im Werkzeug-Fenster mit Hilfe der Tasten [F1] – [F4] verändert werden kann ohne das Werkzeugfenster aktivieren zu müssen!

4.6.2 *Info*

Hier sind Informationen zum Programm zu finden.

4.7 Schalter

Für die wichtigsten Funktionen sind auch Schalter am linken Rand des Schaltplanfensters zu finden.

Unter der Rubrik „Auswerten“ finden sich die Schalter „Berechnen“, „Schritt“, „Takt“, „Zurück“, „Löschen“. Ihre Funktionen entsprechen den Menüpunkten des Menüs „Berechnung“.

Unter der Rubrik „Darstellen“ finden Sich die Schalter „Größer“, „Kleiner“, „Rahmen“. Ihre Funktionen entsprechen den Menüpunkten des Menüs „Ansicht“.

5 Referenzen

[1] www.bkolsberg.de

[2] www.javabuch.de

[3] GoTo Java 2 Handbuch der Java-Programmierung
Guido Krüger
Verlag: Addison-Wesley

6 Anlage: Quellkode

Beispiel für eine „Schaltsymbol-Klasse“ Diese Klasse realisiert die UND-Gatter.
(Andere Quelldateien siehe CD-Rom.)

```
package elektro;
import java.awt.*;

/**
 * Autor:                       BK Olsberg
 * Aufgabe:                     Elektro
*/

public class And extends SymbolBasis
{
        public And (int eingang, boolean ausgang)
        {
                super (eingang, ausgang);
        }

        public void zeichne (Graphics g, int xPixel, int yPixel, RahmenDaten rahmenDaten)
        {
                switch (rahmenDaten.getWert(OSTEN))
                {
                        case NULL:      g.setColor(NULLFARBE);break;
                        case EINS:      g.setColor(EINSFARBE);break;
                        default:        g.setColor(FUELLFARBE);
                }

                super.zeichne(g, xPixel, yPixel, rahmenDaten);

                g.setColor(SCHATTENFARBE_HELL);
                this.und(g, xPixel+1, yPixel+1);
                g.setColor(ZEICHENFARBE);
                this.und(g, xPixel, yPixel);
        }

        public int berechneWert (RahmenDaten rahmenDaten)
        {
                int berechnung = FEHLER;

                if (    (       eingang == 2
                                        &&      (       rahmenDaten.getWert(NORDEN) == EINS
                                                &&      rahmenDaten.getWert(WESTEN) == EINS))
                        ||      (       eingang == 3
                                        &&      (       rahmenDaten.getWert(NORDEN) == EINS
                                                &&      rahmenDaten.getWert(WESTEN) == EINS
                                                &&      rahmenDaten.getWert(SUEDEN) == EINS)))
                {
                        if (ausgang)    berechnung = EINS;
                        else            berechnung = NULL;
                }
                else
                {
                        if (ausgang)    berechnung = NULL;
                        else            berechnung = EINS;
                }


                if (berechnung != FEHLER)
                {
                        rahmenDaten.setWertAusgang(berechnung);
                        rahmenDaten.setFortsetzen (false,false,false,true);
                        rahmenDaten.resetSchaltzeit(true,true,true,true);
                }

                return berechnung;
        }

        public boolean setSchaltzeit(int richtung, RahmenDaten rahmenDaten)
        {
                return rahmenDaten.setSchaltzeit(richtung, Parameter.andSchaltzeit);
        }
}
```