Karlsruher Institut für Technologie
Institut für Technik der Informationsverarbeitung

Dokumentation zum Projektpraktikum Informationstechnik

Gruppe:	57
Gruppenmitglieder:	Sebastian Müller, Kristian Maier, Florian König, Maxim Köhler
Tutor:	Thomas Streitz
Abgabetermin:	28.11.2012
Semester:	WS2012/2013

KIT - Universität des Landes Baden Württemberg und nationales Forschungszentrum in der Helmholtz-Gemeinschaft

Inhaltsverzeichnis

Kapitel 1
Problemstellung

Die Aufgabenstellung war die Analyse eines Schaltwerkes zur Ermittlung der maximalen Betriebsfrequenz. Da jedes elektronische Gatter eine gewisse Laufzeit benötigt, um das Eingangssignal in ein Ausgabesignal zu überführen, muss darauf geachtet werden, dass das nachfolgende Gatter erst getriggert wird, sobald das Eingangssignal korrekt anliegt. Wird zu früh getriggert, so funktioniert das Schaltwerk nicht wie gewollt. Wird zu spät getriggert, geht wertvolle Rechenzeit verloren. Um die Funktionalität des Schaltwerks zu gewährleisten, wird durch unsere Software, die in C++ geschrieben wurde, ein beliebiges Schaltwerk simuliert und die maximale Betriebsfrequenz ermittelt.
Dabei gibt es ein zentrales Menü, über das alle Parameter gesetzt werden können und die Analyse gestartet werden kann. Die Bibliothek möglicher Gatter wird dabei aus einer externen Textdatei importiert und kann somit variiert werden. Das Schaltnetz an sich wird ebenfalls aus einer externen Textdatei eingelesen und ist variabel. Außerdem wird auf gewisse äußere Faktoren Rücksicht genommen: Betriebstemperatur, Versorgungsspannung der Gatter und Eigenschaften durch die Produktion der Gatter.
Eine Beispielausgabe nach dem Starten des Programmes ist im Folgenden zu sehen:

```
****************************************
* IT-Projektpraktikum WS2012/2013 *
* Laufzeitanalyse synchroner Schaltwerke *
****************************************

(1) aeussere Faktoren
Spannung [Volt]: 1.2
Temperatur [Grad Celsius]: 55
Prozess (1=slow, 2=typical, 3=fast): 1

(2) Bibliothek
Pfad zur Bibliotheksdatei: c:\bib.txt

(3) Schaltwerk
Pfad zur Schaltwerksdatei: c:\csd.txt

(4) Analyse starten

(5) Programm beenden


Waehle einen Menuepunkt und bestaetige mit Enter:
```

Ausgabe 1.1: Beispielausgabe beim Starten des Programms

Kapitel 2
Umsetzung

2.1 Gesamtkonzept der Lösung

Alle Attribute der Klassen wurden als `private` oder `protected` angelegt, um kontrollierten Zugriff auf die Attribute über `get-` und `set`-Methoden zu gewährleisten. Außerdem wurden Präprozessordefinitionen verwendet und abgefragt, um je nach Betriebssystem nur den dort funktionierenden Code zu kompilieren. Unter Windows lautet das Präprozessor-Argument *WIN32*, unter Linux/Mac *LINUX*.

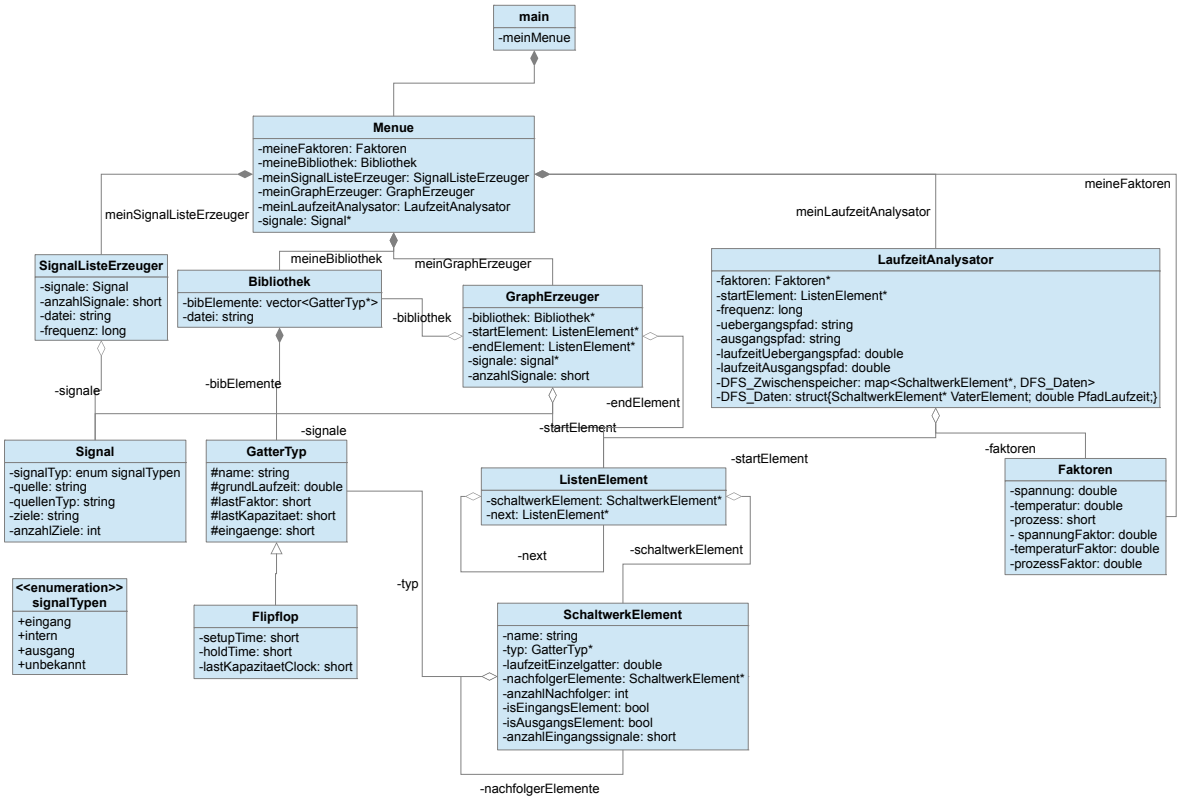

Abbildung 2.1: UML-Diagramm des Gesamtkonzeptes

2.2 Klasse `Menue`

2.2.1 Beschreibung

Mithilfe der Klasse `Menü` kann der Benutzer den gesamten Programmablauf steuern. Im Hauptmenü kann mittels `switch-case` Abfrage in die Untermenüs navigiert werden, in denen der Spannungswert, die Temperatur, der Prozessfaktor eingetragen sowie der Pfad zur Bibliotheks- und Schaltnetzdatei festgelegt werden kann. Bei ungültigen Eingaben wird der Nutzer durch eine Fehlermeldung darauf hingewiesen. Außerdem lässt sich die Laufzeitanalyse, bei der im Menü direkt überprüft wird, ob die angelegte Taktfrequenz kleiner oder gleich der, durch die Analyse, berechneten Frequenz ist, direkt aus dem Hauptmenü heraus starten. Desweiteren kann der Nutzer vordefinierte Testwerte/-pfade direkt einfügen, ohne erst mühsam die einzelnen Parameter und Pfade eintragen zu müssen. Ebenfalls lässt sich das Programm im Hauptmenü beenden.

2.3 Klasse `Bibliothek`

2.3.1 Beschreibung

Die Klasse Bibliothek dient dazu, Informationen zu den vorhandenen Gatter-Typen auf Basis einer Bibliotheksdatei zur Verfügung zu stellen. Dazu wird zunächst mit Hilfe der Methode `pfadEinlesen()` ein gültiger Pfad zu einer vorhandenen Datei übergeben, der im Menü abgefragt wird. Daraufhin kann diese wahlweise mit der Methode `dateiAusgabe()` ausgegeben werden oder man bereitet die weitere Verarbeitung mit der Methode `dateiAuswerten()` vor.

In der Bibliotheksdatei sind zunächst alle vorhandenen Bauteile unter der Zeile "[[Bauteile]]" aufgelistet. Daraufhin folgen nach einer Leerzeile die genauen Daten jedes Bauteils. Ein solcher Datensatz startet mit der einleitenden Zeile "[*Kurzbezeichnung des Gatters*]", auf die je eine Zeile pro Attribut des Gatters folgt. Diese Zeile sind nach dem Schema "*Name*: *Wert*" aufgebaut.

Nachdem die Datei ausgewertet wurde, können die Daten eines Gattertyps komfortabel anhand des Namen als String mit der Methode `getBibElement()` abgerufen und in anderen Programmteilen verwendet werden.

2.3.2 Test der Klasse

In der Testfunktion `bib_testing()` wird zunächst der Pfad eingelesen und daraufhin die Datei ausgegeben und ausgewertet. Danach werden die Daten jedes Gattertyps mit der Methode `getBibElement()` abgerufen und mit der Methode `out()` der Klasse `GatterTyp`, die nur zu Testzwecken erstellt wurde, und einfach sämtliche Daten des Objekts ausgibt, auf der Konsole sichtbar gemacht. Das Flipflop wird dabei gesondert behandelt, da es durch mehr Daten charakterisiert wird.

Die Funktion ist im Folgenden zu sehen:

Listing 2.1: Code der Testfunktion zur Klasse `Bibliothek`

```
1  void bib_testing(){
2      Bibliothek bib;
3      bib.pfadEinlesen();
4      bib.dateiAusgabe();
5      bib.dateiAuswerten();
6      GatterTyp* tempGatter;
7      cout << endl;
8      string title[9] = {"inv1a", "inv1b", "inv1c", "and2", "or2", "↩
           nand2", "nor2", "xor2", "xnor2"};
9      for( int i = 0; i < 9; i++) {
10         tempGatter = bib.getBibElement(title[i]);
11         tempGatter->out();
12     }
13     tempGatter = bib.getBibElement("dff");
14     dynamic_cast<Flipflop*>(tempGatter)->out();
15     string moin;
16     cin >> moin;
17 }
```

Der Test der Klasse verlief erwartungsgemäß. Die Ausgabe ist im Folgenden zu sehen:

```
0: //-------------------------
1: //Bibliothek zu Projektpraktikum Informationstechnik
2: //Autor: Tobias Schwalb & Adnene Gharbi, ITIV
3: //Erstellt: 30.05.2008
4: //Version: 1.1
5: //-------------------------
6:
7: begin
8: [[Bausteine]]
9: inv1a
10: inv1b
11: inv1c
...
//Weitere Zeilen aus Platzgründen weggelassen
...
GatterTyp 'inv1a':
Grundlaufzeit: 26.5
Lastfaktor: 800
Lastkapazitaet: 7
Eingaenge: 1

GatterTyp 'inv1b':
Grundlaufzeit: 26.5
Lastfaktor: 300
Lastkapazitaet: 13
Eingaenge: 1
...
//Weitere Zeilen aus Platzgründen weggelassen
```

Ausgabe 2.1: Ausgabe der Bibliotheks-Testfunktion

2.4 Klasse `GatterTyp` und `Flipflop`

2.4.1 Beschreibung

Ein Objekt der Klasse `GatterTyp` speichert alle notwendigen Informationen zu einem bestimmten Gattertyp. Die Klasse `Bibliothek` verwaltet alle vorhandenen Gattertypen und ermöglicht den anderen Programmbestandteilen den Zugriff auf die Gattertypen mit der Methode `getBibElement()`. Da die Klasse `GatterTyp` im Prinzip nur als Datenspeicher dient, besteht sie auch fast ausschließlich aus `get-` und `set-`Methoden.
Da Flipflops durch mehr Daten charakterisiert sind als die anderen Gattertypen, wird für Flipflops konsequenterweise eine eigene Klasse definiert, die eine Kindklasse von `GatterTyp` ist. Diese besteht ebenso überwiegend aus `get-` und `set`-Methoden. Eine besondere Rolle nimmt die Methode `getIsFlipflop()` ein. Sie ist als `virtual` definiert, sodass mit Hilfe dieser Funktion in anderen Programmteilen überprüft werden kann, ob es sich um ein Flipflop oder ein anderes Gatter handelt.

2.4.2 Test der Klassen

Da die Klassen keine große Funktionalität außer der Speicherung von Daten besitzen, können sie einfach getestet werden, indem ein Testobjekt erstellt wird, dessen Attribute über die `set`-Methoden Werte zugewiesen bekommen. Daraufhin werden alle Attribute über die `get`-Methoden ausgelesen und auf der Konsole ausgegeben. Aufgrund der Trivialität dieses Verfahrens soll hier nicht weiter darauf eingegangen werden.

2.5 Klasse `Faktoren`

2.5.1 Beschreibung

Die Klasse `Faktoren` übernimmt die Berechnung spezifischer Konstanten, die sich aus den drei äußeren Faktoren Versorgungsspannung (VDD), Betriebstemperatur (Temp.) und Variationen bei der Herstellungen (Prozess) ergeben.

Die Gatterlaufzeit eines Bauelements ist proportional zu den drei errechneten Konstanten. Die Versorgungsspannung hat einen gültigen Bereich von $1,08$ V bis $1,32$ V. Der Zugehörige Faktor K_v nimmt mit zunehmender Spannung ab. Die Betriebstemperatur kann zwischen $-25°C$ und $125°C$ gewählt werden, der zugehörige Faktor K_t nimmt dabei mit steigender Temperatur linear zu. Für den Prozess gibt es drei fest vorgegebene Einstellmöglichkeiten: `slow`, `typical` und `fast`. Der Faktor K_p nimmt dabei linear ab.

Zur Berechnung der Faktoren K_v und K_t wird auf eine Tabelle zugegriffen, die als zweidimensionales Array implementiert wurde. Befindet sich die gesetzte Größe in dieser Tabelle, gibt die Methode `berechneFaktor()` den Faktor direkt zurück. Befindet sich die gesetzte Größe im gültigen Bereich, aber nicht in der Tabelle, so wird mittels der Methode `interpolation()` ein Mittelwert zwischen den zwei nächsten Werten in der Tabelle errechnet. Befindet sich die Größe nicht im gültigen Bereich, wird ein Fehler geworfen.

2.5.2 Test der Klasse

Zum Testen der Faktoren-Klasse wurde eine Funktion geschrieben, in der die drei äußeren Größen per Benutzereingabe abgefragt wurden. Danach wurden die Eingaben in die Attribute der Faktoren-Instanz geschrieben und alle Methoden zum Testen aufgerufen.

Der Test verlief wie erwartet, war ein Wert aus der Tabelle eingegeben worden, so wurde der zugehörige Faktor ausgegeben. War der Wert nicht in der Tabelle, wurde ein realistischer Zwischenwert ausgegeben. Bei einer Eingabe außerhalb der erlaubten Größen wurde eine Fehlermeldung ausgegeben.

Listing 2.2: Code der Testfunktion zur Faktoren-Klasse

```
1  void faktoren_testing() {
2      double kv, kt, kp, sp, tm, pr;
3      Faktoren faktoren;
4      cout << "Spannung? ";
5      cin >> sp;
6      cout << "Temperatur? ";
7      cin >> tm;
8      cout << "Prozess? ";
9      cin >> pr;
10     faktoren.setProzess(pr);
11     cout << "setProzess() ausgefuehrt." << endl;
12     faktoren.setSpannung(sp);
13     cout << "setSpannung() ausgefuehrt" << endl;
14     faktoren.setTemperatur(tm);
15     cout << "setTemperatur() ausgefuehrt." << endl;
16     cout << "getProzess() ausgefuehrt: " << faktoren.getProzess() << ↩
           endl;
17     cout << "getSpannung() ausgefuehrt: " << faktoren.getSpannung() <<↩
            endl;
18     cout << "getTemperatur() ausgefuehrt: " << faktoren.getTemperatur↩
           () << endl;
19     cout << "ausgabeFaktoren() wird jetzt ausgefuehrt:" << endl;
20     faktoren.ausgabeFaktoren();
21     faktoren.getFaktoren(kv, kt, kp);
22     cout << "getFaktoren() Ergebnis: KV=" << kv << " , KT=" << kt << "↩
            , KP=" << kp << endl;
23 }
```

```
Spannung? 1.1
Temperatur? 37
Prozess? 2
setProzess() ausgefuehrt.
setSpannung() ausgefuehrt
setTemperatur() ausgefuehrt.
getProzess() ausgefuehrt: 2
getSpannung() ausgefuehrt: 1.1
getTemperatur() ausgefuehrt: 37
ausgabeFaktoren() wird jetzt ausgefuehrt:
-----------------
Faktoren: KV: 1.09844 | KT: 1.02435 | KP: 1
-----------------
getFaktoren() Ergebnis: KV=1.09844 , KT=1.02435 , KP=1
```

Ausgabe 2.2: Ausgabe der Faktoren-Testfunktion

2.6 Klasse `Signal` und `SignalListeErzeuger`

2.6.1 Beschreibung

Die Klasse `Signal` beschreibt ein Signal zwischen 2 Gattern des Schaltwerks. Wie in Abbildung 2.1 zu sehen ist, sind alle Attribute private. Der Zugriff erfolgt über `get-` und `set-` Methoden. Eine Ausnahme ist das Attribut `ziele`, da es ein Array ist. Der schreibende Zugriff erfolgt hier über die Methode `zielHinzufuegen()`. Der Konstruktor initialisiert alle Attribute mit 0, bzw. Strings mit einem leeren String("").
Die Hauptaufgabe der Klasse `SignalListeErzeuger` ist die Erzeugung einer Liste von Signalen. Diese wird in Form eines dynamischen Arrays aus Objekten der Klasse `Signal` realisiert. Sie wird benötigt, um später daraus einen Graphen aufzubauen, der Knoten die Gatter und Kanten die Signale sind.
Die Informationen zum Schaltwerk sind in der sogenannten Schaltnetzdatei gespeichert. Deren Aufbau ist genau vorgegeben und in 3 Blöcke unterteilt .Im ersten Block ist der Name des Schaltnetzes und des Projektes gespeichert. Im sogenannten ENTITY-Kopf werden die Signale mit ihrem entsprechenden Typ(Eingang-, Ausgang- oder internes Signal) aufgezählt. Auch die Frequenz des Clock-Signals wird dort angegeben. Im letzten Block werden die Gatter aufgezählt und mit welchen Signalen sie verknüpft sind.
Die Erzeugung der Liste wird mit der Methode `Signal* erzeugeSignalliste()` begonnen, die das Problem in mehreren Teilschritten löst. Dazu werden mehrere private Methoden implementiert, um den Prozess übersichtlicher zu gestalten. Zunächst wird der aus der Schaltnetzdatei, deren Pfad vorher an die Klasse übergeben werden muss, der ENTITY-Kopf ausgelesen. Dies erledigt die Methode `readEntity()`. Mit `countSignals()` werden anschließend die Anzahl aller Signale ermittelt, damit im nächsten Schritt das dynamische Array `signale` erzeugt werden kann. Anschließend übernimmt `readFrequenz()` das Auslesen der Frequenz. Mit der Methode `fillList()` werden den in der Liste vorhandenen Signalen ihren Typ zugeordnet. Dabei entspricht die Position im Array der Signalnummer wie sie in der Schaltnetzdatei steht und kann so weiter eindeutig zugeordnet werden. Zuletzt wird mit `readGatterInfo()` aus der Datei ausgelesen, mit welchen Gattern die Signale verbunden sind. Auch diese Information wird dann in der Liste abgespeichert. Weiterhin gibt die Methode zurück, ob im Schaltnetz ein Kurzschluss auftritt, d.h. ob ein interner Ausgang eines Gatters mit einem anderen internen Ausgang eines anderen Gatters oder mit einem externen Eingang verbunden ist (den dieser ist mit dem Ausgang eines externen Gatters verbunden).
Der oben beschriebene Prozess ist dabei mit einem `try-catch` Block umrundet, um Fehler abzufangen. Ist z.B. die Schaltnetzdatei fehlerhaft und kann nicht ausgelesen werden, stürzt das Programm nicht ab, sondern es wird er `NULL`-Zeiger zurückgegeben.

2.6.2 Test der Klasse

Zum Testen der Klasse wurde eine eigene Methode entworfen. Diese Methode fragt zunächst vom Benutzer den Pfad der Schaltnetzdatei ab. Ist diese Datei auslesbar, wird die Schaltnetzdatei ausgeben, die Signalliste erzeugt und abschließend die Signalliste ausgeben.

Listing 2.3: Testfunktion der `SignalListeErzeuger`-Klasse

```
1 void signal_testing(){
2     SignalListeErzeuger s;
3     cout << "Pfad zur Schaltnetzdatei:" << endl;
```

```
4       string buf;
5       getline(cin, buf);
6
7       if(s.setPfadSchaltnetzdatei(buf)) {
8           s.ausgabeSchaltnetzdatei();
9           Signal* sig = s.erzeugeSignalliste();
10          if(sig != NULL){
11              s.ausgabeSignale();
12          }
13      }else{
14          cout << "Falscher Pfad" << endl;
15      }
16  }
```

Der Test verlief erwartungsgemäß. Bei fehlerhafter Eingabe des Pfades wurde die Funktion beendet, sonst wurde die Schaltnetzdatei ausgegeben, die Signalliste erzeugt und falls dies erfolgreich war die Liste ausgeben. Auch der Test mit einem kurzschlussbehafteten Schaltwerk ergab das gewünschte Resultat.

```
Pfad zur Schaltnetzdatei:
csd.txt
//Hier die Ausgabe des Dateiinhaltes, wurde aus Platzgründen weggelassen
Signale:
---
Signalname : s001
Signaltyp : Eingangssignal
Signalquelle : keine Quelle
-> Das Signal hat 2 Ziele
Ziel-Gatter : g015 g002
---
Signalname : s002
Signaltyp : Internes Signal
Signalquelle : g002
-> Das Signal hat 2 Ziele
Ziel-Gatter : g008 g004
---
...
//Weitere Signale wurden aus Platzgründen weggelassen
```

Ausgabe 2.3: Ausgabe der Testfunktion (gekürzt)

2.7 Klasse `GraphErzeuger`, `ListenElement` und `SchaltwerkElement`

2.7.1 Beschreibung von `ListenElement` und `SchaltwerkElement`

Die Klasse `ListenElement` dienst zum implementieren einer einfach verketteten Liste, deren Elemente Instanzen dieser Klasse sind. Sie verfügt über nur zwei Attribute: einen Zeiger auf ein Objekt vom Typ `SchaltwerkElement` und einen Zeiger auf den Nachfolger in der Liste.
Die Klasse `SchaltwerkElement` repräsentiert ein Gatter und hat dem entsprechend die Attribute Name, Gattertyp, Laufzeit, Nachfolger, Nachfolgeranzahl, Anzahl der Eingangssignale und die Information, ob es sich um ein Eingangs- oder Ausgangselement handelt. Neben den üblichen `get`- und `set`-Methoden verfügt die Klasse noch über eine Methode `NachfolgerHinzufuegen(SchaltwerkElement* schaltwerkElement, int pos)`, die ein Nachfolgergatter hinzufügt, also einen Zeiger auf einen Nachfolger in das Array der Nachfolger einträgt, und die Anzahl der Nachfolger um eins erhöht.

2.7.2 Beschreibung von `GraphErzeuger`

Die Klasse `GraphErzeuger` erzeugt den Graphen, d.h. die Repräsentation der Verbindungen zwischen den Gattern des Schaltwerks.
Dazu wird zuerst eine einfach verkettete Liste erstellt, die Elemente vom Typ `ListenElement` enthält. Zum erstellen der Liste wird die Signalliste des `SignalListeErzeuger` durchgegangen und für jede Quelle (sofern es sich um kein Eingangssignal handelt) ein `ListenElement` in die Liste eingefügt, das mit einem ebenfalls

neu erstellten `SchaltwerkElement` verknüpft wird. Dabei werden schon die aus der Signalliste bekannten Attribute Gattertyp, Name und die Information, ob es sich um ein Eingangs- oder Ausgangselement handelt, im `SchaltwerkElement` gespeichert.

Danach wird der Graph erzeugt, indem ein weiteres mal alle Signale durchgegangen werden. Verfügt das Signal über mindestens ein Ziel, so wird die Anzahl der Eingangssignale aller Zielgatter um eins erhöht. Verfügt das Signal außerdem noch über eine Quelle, so werden alle Ziele mit dem Quellgatter verknüpft, indem über die Methode `NachfolgerHinzufuegen()` die Ziele als `SchaltwerkElement` übergeben werden. Zum Finden der richtigen Zeiger auf die bereits existierenden Nachfolger wird die private Methode `sucheGatter(string Gatter)` benutzt, die eine lineare Suche durch alle `SchaltwerkElemente` nach dem Namen durchführt und einen Zeiger auf das richtige Objekt zurück gibt.

Zu diesem Zeitpunkt sind fast alle Attribute aller `SchaltwerkElemente` korrekt gesetzt: Name, Gattertyp, Nachfolger, Nachfolgeranzahl, Eingangs- oder Ausgangselement und die Anzahl der Eingangssignale. Was fehlt ist die Laufzeit des Einzelgatters, die erst durch der Klasse `LaufzeitAnalysator` eingetragen wird.

Abschließend wird noch überprüft, ob es ungenutzte Signale gibt, d.h. falls ein internes oder Eingangssignal in kein Gatter führt. Außerdem wird geprüft, ob die Anzahl der Eingangssignale jedes Gatters mit der Definition des Bauteils in der Bibliothek übereinstimmt.

2.7.3 Test der Klasse

Zum Testen der Klasse wurde eine Testfunktion erstellt. Zuerst wird dort eine Instanz der Klasse `Bibliothek` erzeugt und eine Bibliotheksdatei eingelesen. Danach wird eine Instanz der Klasse `SignalListeErzeuger` erstellt und dort ebenfalls die Schaltnetzdatei eingelesen. Die Bibliothek und die Signalliste werden an eine Instanz des `GraphErzeugers` übergeben.

Danach wird die Methode `erzeugeGraph()` aufgerufen und die Schritte aus der Beschreibung der Klasse ausgeführt. Abschließend wird die Methode `ausgabeGraph()` aufgerufen, um den Graphen in der Konsole auszugeben. Die Tests verliefen wie erwartet und erfolgreich.

Listing 2.4: Testfunktion der GraphErzeuger-Klasse

```
1  void graph_testing() {
2      GraphErzeuger graph;
3      Bibliothek bib;
4      bib.pfadEinlesen();
5      bib.dateiAuswerten();
6      graph.setBibliothek(&bib);
7
8      SignalListeErzeuger s;
9      cout << "Pfad zur Schaltnetzdatei:" << endl;
10     string buf;
11     cin >> buf;
12     bool ok;
13     ok = s.setPfadSchaltnetzdatei(buf);
14     if(!ok){
15         cout << "Falscher Pfad" << endl;
16     }
17     Signal* sig = s.erzeugeSignalliste();
18     graph.setAnzahlSignale(s.getAnzahlSignale());
19     graph.setSignale(sig);
20
21     graph.erzeugeGraph();
22     graph.ausgabeGraph();
23 }
```

Eine Ausgabe dieser Funktion ist im Folgenden zu sehen.

```
Pfad und Name der Bibliothekdatei eingeben (Abbruch='EXIT'): bib.txt

Pfad erfolgreich eingetragen!

Pfad zur Schaltnetzdatei:
csd.txt

Gatterame: g002
Gattertyp: and2
->Das Gatter hat 2 Ziele
Angeschlossene Gatter : g004 g008
----------------
Gatterame: g004
Gattertyp: and2
->Das Gatter hat 2 Ziele
Angeschlossene Gatter : g003 g013
----------------
...
//Weitere Gatter werden aus Platzgründen weggelassen
```

Ausgabe 2.4: Ausgabe der Testfunktion (gekürzt)

2.8 Klasse LaufzeitAnalysator

2.8.1 Beschreibung

Die Klasse `LaufzeitAnalysator` ermittelt anhand des zuvor erzeugten Graphen und der Laufzeit beeinflussenden Faktoren die Signallaufzeit des längsten Pfades innerhalb des Schaltwerks. Dies geschieht mithilfe eines Tiefensuchalgorithmus. Dazu wird zwischen dem Ausgabeschaltnetz und dem Überführungsschaltnetz unterschieden. Das Ausgabeschaltnetz ist bestimmt durch die längste Verzögerungszeit von einem externen Eingang oder Ausgang eines Flipflops zu einem externen Ausgang. Beim Überführungsschaltnetz ist der Startpunkt auch ein externer Eingang oder der Ausgang eines Flipflop, das Ziel jedoch der Eingang eines Flipflop.
Bevor die Laufzeitanalyse mit der Methode `start_LZA()` gestartet werden kann, müssen die Objekte der Klassen `GraphErzeuger` und `Faktoren` übergeben werden, die das Schaltnetz beschreiben. Dazu existieren `set`-Methoden. Zur Vorbereitung der Tiefensuche wird zunächst die Laufzeit der Einzelgatter berechnet. In diese fließen die Faktoren (Spannung, Temperatur und Prozess), die Grundlaufzeit, die Lastkapazität und der Fan-Out der Gatter ein. Nun wird die Tiefensuche für jedes Gatter aufgerufen, welches ein gültiges Startelement ist (Eingangselement oder Flipflop). Dazu wird die Methode `dfs()` aufgerufen, die zunächst die Startwerte initialisiert und `dfs_visit()` aufruft, wo der eigentliche Tiefensuchalgorithmus implementiert ist. Die Tiefensuche verläuft rekursiv und arbeitet in die Tiefe, d.h. es werden zuerst die Folgeknoten des Graphen betrachtet, bevor die „Nachbarn“ dran sind. Ist ein Folgeknoten ein Flipflop, so ist ein Endelement des Übergangpfades gefunden und eine neue möglicherweise größere Laufzeit des Übergangpfades ist gefunden. Wenn sie größer ist, wird sie abgespeichert.
In einem weiteren Schritt wird auch eine Zyklensuche durchgeführt. Ein möglicher Zyklus ergibt sich, wenn das Folgeelement Teil eines bereits bearbeiteten Pfades ist. Ein Zyklus ist ein Fehler im Schaltnetz, was zum Abbruch des Algorithmus und Ausgabe einer Fehlermeldung führt. Die Zyklusüberprüfung geschieht mit der Methode `zyklensuche()`. Im letzten Schritt der Tiefensuche wird geprüft, ob der aktuelle Knoten Teil mit einem Ausgangssignal verbunden ist. Dann wäre ein neuer Ausgangspfad gefunden.

Listing 2.5: Die Methode `dfs_visit()`

```
1 bool LaufzeitAnalysator::dfs_visit(SchaltwerkElement* k, ↩
    SchaltwerkElement* s, string pfad){
2     SchaltwerkElement* v;
3     double tempZeit;
4
5     //verfolgter Pfad speichern
6     pfad += "->" + k->getName();
7
8     //alle Nachfolger iterieren
```

```
9       for(int i = 0; i < k->getAnzahlNachfolger(); i++){
10          v = k->getNachfolger(i);
11          tempZeit = DFS_Zwischenspeicher[k].pfadLaufzeit + k->↩
                getLaufzeitEinzelgatter();
12
13          //wenn Flipflop ist Endelement des Uebergangpfades gefunden
14          if(v->getTyp()->getIsFlipflop()){
15              //wenn neue Laufzeit groesser, neue Maximale Laufzeit ↩
                    setzen
16              if(laufzeitUebergangspfad < tempZeit){
17                  laufzeitUebergangspfad = tempZeit;
18                  uebergangspfad = pfad + "->" + v->getName();     //↩
                        Pfadstring der groessten Laufzeit speichern
19                  Flipflop* ff = (Flipflop*)(v->getTyp());         //↩
                        Referenz auf den FlipFloptyp
20                  //berechne Frequenz in Hz
21                  frequenz = (long) ( 1e15 / ( laufzeitUebergangspfad + ↩
                        1000 * ff->getSetupTime()) );
22              }
23          //sonst wenn moeglicher laengerer Pfad
24          }else if(DFS_Zwischenspeicher[v].pfadLaufzeit < tempZeit){
25              //möglicher Zyklus pruefen
26              if(((DFS_Zwischenspeicher[v].pfadLaufzeit != 0) || (v == s↩
                    )) &&
27                      DFS_Zwischenspeicher[v].vaterElement != k){
28
29                      DFS_Zwischenspeicher[v].vaterElement = k;
30                      if(zyklensuche(v)){
31                          return false;
32                      }
33              }
34              //setzen der Werte des Folgeknotens und rekursiver Aufruf ↩
                    der dfs
35              tempZeit = DFS_Zwischenspeicher[k].pfadLaufzeit + k->↩
                    getLaufzeitEinzelgatter();
36              DFS_Zwischenspeicher[v].pfadLaufzeit = tempZeit;
37              DFS_Zwischenspeicher[v].vaterElement = k;
38              if(!dfs_visit(v, s, pfad)){
39                  //wenn Zyklus gefunden
40                  return false;
41              }
42
43          }
44      }
45
46      tempZeit = DFS_Zwischenspeicher[k].pfadLaufzeit + k->↩
            getLaufzeitEinzelgatter();
47
48      //wenn Knoten mit Ausgang verbunden ist, ist neuer Ausgangspfad ↩
            gefunden
49      if(k->getIsAusgangsElement() && laufzeitAusgangspfad < tempZeit){
50          laufzeitAusgangspfad = tempZeit;
51          ausgangspfad = pfad;
52      }
53
54      return true;
55
56  }
```

2.8.2 Test der Klasse

Zum Testen der Laufzeitanalyse wurde die Menüführung benutzt. Fort werden die äußeren Faktoren, der Pfad zur Schaltnetzdatei und Bibliotheksdatei gesetzt. Anschließend wird die Laufzeitanalyse gestartet. Getestet wurden mehrere Schaltwerke, die auch fehlerbehaftet sind (offener Eingang, unbenutztes Signal, Zyklus, Kurzschluss, falsche Anzahl von Eingangssignalen). Alle Fehler wurden vom Programm erkannt und gemeldet, auch für das Schaltnetz mit dem Zyklus, dessen Überprüfung innerhalb der Laufzeitanalyse erfolgt. Für ein fehlerfreies Schaltnetz wurden die korrekten Laufzeiten der Pfade ausgegeben.
Eine Beispielausgabe einer erfolgreichen LZA ist in Kapitel 3 zu finden, hier eine Ausgabe bei einem gefunden Zyklus:

```
****************************************
* IT-Projektpraktikum WS2011/2012 *
* Laufzeitanalyse synchroner Schaltwerke *
****************************************


(1) aeussere Faktoren
Spannung [Volt]: 1.2
Temperatur [Grad Celsius]: 55
Prozessor (1=slow, 2=typical, 3=fast) : 1


(2) Bibliothek
Pfad zur Bibliotheksdatei: bib.txt


(3) Schaltwerk
Pfad zur Schaltwerksdatei: test_Zyklus_1.txt


(4) Analyse starten


(5) Testwerte einfuegen


(6) Programm beenden



Waehle einen Menuepunkt und bestaetige mit Enter: 4

Fehler! Zyklus an g004 gefunden!

LZA Fehlgeschlagen!

Enter drücken...
```

Ausgabe 2.5: Abbruch der Analyse bei Zyklus

2.9 Hauptprogramm

Das Hauptprogramm, dessen Quelltext sich in der Methode `main()` findet, stellt gleichzeitig die Testumgebung für die Klasse `Menue` dar. Die einzige Aufgabe dieser Methode ist das Erstellen einer Instanz der Klasse `Menue` und das Aufrufen der Methode `start()` der Instanz. Daraufhin wird das Programm mit der Anzeige des Menüs gestartet.

Listing 2.6: Quelltext des Hauptprogramms

```
1 /*
2  * Datei:   main.cpp
```

```
3  * Author: Sebastian Müller, Kristian Maier, Maxim Köhler, Florian ↩
       König
4  *
5  * IT Praktikum WS 2012/13
6  * Gruppe 57
7  */
8
9
10 #include <iostream>
11 #include "Menue.h"
12
13 using namespace std;
14
15 int main() {
16     Menue menue;
17     menue.start();
18     return 0;
19 }
```

Kapitel 3
Abschlusstest

Zum Testen des Gesamtprogrammes nutzten wir alle bereitgestellten Schaltwerksdateien, die mit verschiedenen Fehlern ausgestattet waren. Außerdem erstellten wir noch selbst einige fehlerhafte Schaltwerksdateien. Alle Fehler wurden von unserem Programm erkannt und eine Fehlermeldung wurde ausgegeben (s. z.B. 2.8.2).
Außerdem wendeten wir das Prinzip des DAU (*Dümmster anzunehmender User*) an, in dem wir mögliche Bedienfehler des Programms testeten.
Hier die Ausgabe des fertigen Programms mit den bereitgestellten Bibliotheks- und Schaltnetzdateien bei einer Versorgungsspannung von 1,2 V, einer Temperatur von 55°C und dem Prozesstyp `slow`:

```
****************************************
* IT-Projektpraktikum WS2012/2013 *
* Laufzeitanalyse synchroner Schaltwerke *
****************************************


(1) aeussere Faktoren
Spannung [Volt]: 1.2
Temperatur [Grad Celsius]: 55
Prozessor (1=slow, 2=typical, 3=fast) : 1

(2) Bibliothek
Pfad zur Bibliotheksdatei: bib.txt

(3) Schaltwerk
Pfad zur Schaltwerksdatei: csd.txt

(4) Analyse starten

(5) Testwerte einfuegen

(6) Programm beenden



Waehle einen Menuepunkt und bestaetige mit Enter: 4

Laengster Pfad im Ueberfuehrungsschaltnetz:
->g014->g002->g004->g013->g011->g001
Maximale Laufzeit der Pfade im Ueberfuehrungsschaltnetz: 664.129 ps

-------------------------
Laengster Pfad im Ausgangsschaltnetz:
->g014->g002->g004->g013->g007
Maximale Laufzeit der Pfade im Ausgangsschaltnetz: 602.909 ps

-------------------------
Maximale Frequenz: 1449 Mhz


Enter drücken...
```

Ausgabe 3.1: Ausgabe des fertigen Programms

Listing 3.1: Inhalt der Datei `bib.txt`

```
//-----------------------------------------------------
//Bibliothek zu Projektpraktikum Informationstechnik
//Autor: Tobias Schwalb & Adnene Gharbi, ITIV
//Erstellt: 30.05.2008
//Version: 1.1
//-----------------------------------------------------

#begin
[[Bausteine]]
inv1a
inv1b
inv1c
and2
or2
nand2
nor2
xor2
xnor2
dff

[inv1a]
ei:1
tpd0:26.5
kl:800
cl:7

[inv1b]
ei:1
tpd0:26.5
kl:300
cl:13

[inv1c]
ei:1
tpd0:87.5
kl:200
cl:7

[and2]
ei:2
tpd0:66.0
kl:2950
cl:3

[or2]
ei:2
tpd0:70.0
kl:1700
cl:3

[nand2]
ei:2
tpd0:38.5
kl:6100
cl:3

[nor2]
```

```
ei:2
tpd0:75.5
kl:800
cl:3

[xor2]
ei:2
tpd0:111.5
kl:1200
cl:4

[xnor2]
ei:2
tpd0:98.5
kl:3100
cl:4

[dff]
ed:1
tsetup:26
thold:30
cd:3
et:1
tpdt:120.5
kl:2650
ct:3
#endfile
}
```

Listing 3.2: Inhalt der Datei `csd.txt`

```
// Diese Schaltnetzdatei enthält einen
// kaskadierenden Rückwertzzähler

ARCHITECTURE schaltwerk2 OF Informationstechnik IS

ENTITY
INPUT s001;
OUTPUT s005,s014,s015,s016,s017;
SIGNALS s002,s003,s004,s006,s007,s008,s009,s010,s011,s012,s013;
CLOCK clk, 1500 MHz;

BEGIN
g001:dff(s013,clk,s009);
g002:and2(s001,s006,s002);
g003:xor2(s003,s008,s012);
g004:and2(s002,s007,s003);
g005:inv1a(s006,s014);
g006:dff(s011,clk,s007);
g007:and2(s004,s009,s005);
g008:xor2(s002,s007,s011);
g009:inv1a(s009,s017);
g010:inv1a(s008,s016);
g011:xor2(s004,s009,s013);
g012:inv1a(s007,s015);
g013:and2(s003,s008,s004);
g014:dff(s010,clk,s006);
g015:xor2(s001,s006,s010);
g016:dff(s012,clk,s008);
```

```
29 END
```

Die Werte unserer Ausgabe stimmten mit der Beispielausgabe in den Projektunterlagen überein.

Anhang A
Quelltext

A.1 Klasse Menue

Listing A.1: Inhalt der Datei `Menue.h`

```
1  /*
2   * Datei:  Menue.h
3   * Author: Florian König
4   *
5   * IT Praktikum WS 2012/13
6   * Gruppe 57
7   */
8
9  #ifndef MENUE_H
10 #define MENUE_H
11
12 #include <iostream>
13 #include <string>
14 #include <stdlib.h>
15
16 #include "Faktoren.h"
17 #include "Bibliothek.h"
18 #include "SignalListeErzeuger.h"
19 #include "GraphErzeuger.h"
20 #include "LaufzeitAnalysator.h"
21 #include "Signal.h"
22
23 using namespace std;
24
25 class Menue {
26 private:
27     Faktoren meineFaktoren;
28     Bibliothek meineBibliothek;
29     SignalListeErzeuger meinSignalListeErzeuger;
30     GraphErzeuger meinGraphErzeuger;
31     LaufzeitAnalysator meinLaufzeitAnalysator;
32     Signal* signale;
33
34     bool spannung_gesetzt;
35     bool temperatur_gesetzt;
36     bool prozess_gesetzt;
37     bool bibliothekspfad_gesetzt;
38     bool schaltnetzpfad_gesetzt;
39
40     void pause();
41     void screenLoeschen();
42     void clear_cin();
43
44     void faktorenMenue();
45     void bibliothekMenue();
46     void schaltwerkMenue();
47     void analyse();
48     void menueKopf();
```

```
49      void testWerteEinfuegen();
50      bool checkEingabe(string str);
51
52  public:
53      Menue();
54      ~Menue();
55      void start();
56
57
58
59  };
60  #endif  /* MENUE_H */
```

Listing A.2: Inhalt der Datei Menue.cpp

```
1   /*
2    * Datei:  Menue.cpp
3    * Author: Florian König
4    *
5    * IT Praktikum WS 2012/13
6    * Gruppe 57
7    */
8
9
10  #include <fstream>
11
12  #include "Menue.h"
13
14  using namespace std;
15
16  /*
17   * Die nachfolgenden Attribute dienen zur Kontrolle, ob jeder für die ↩
         Analyse benötigte
18   * Wert/Pfad gesetzt wurde. Der Konstruktor initialisiert alle ↩
         Atrribute mit false
19   */
20  Menue::Menue() {
21      bool spannung_gesetzt = false;
22      bool temperatur_gesetzt = false;
23      bool prozess_gesetzt = false;
24      bool bibliothekspfad_gesetzt = false;
25      bool schaltnetzpfad_gesetzt = false;
26  }
27
28  // Destruktor - Tut nichts
29  Menue::~Menue() { }
30
31   /*
32    * Diese Methode wartet so lange, bis eine Eingabe getaetigt wuirde.
33    * Preprozessor Abfrage, ob auf Mac oder Windows kompiliert wird.
34    */
35  void Menue::pause() {
36      #if defined( WIN32 )
37          system( "pause" );
38      #endif
39      #if defined( LINUX )
40      cout << endl << "Enter drücken..." << endl;
41      clear_cin();
42      #endif
```

```
43 }
44
45 // Diese Methode leert das Kommando-Fenster
46 void Menue::screenLoeschen() {
47     //Preprozessor Abfrage, ob auf Mac oder Windows kompiliert wird.
48      #if defined( WIN32 )
49         system( "cls" );
50      #endif
51      #if defined( LINUX )
52         system( "clear" );
53      #endif
54 }
55
56 /*
57  * Diese Methode gibt das Hauptmenue auf dem Bildschirm aus.
58  * Durch Eingabe einer Zahl (1-6) gelangt man in das jeweilige Untermenue,
59  * kann die Analyse starten, Testwerte einfügen oder das Programm beenden
60  */
61 void Menue::start() {
62     //Solange bis return kommt
63     while( true ) {
64         int menuepunkt = 0;
65         string menuepunktstring = "";
66
67         screenLoeschen();
68         //Gebe Menuekopf aus
69         menueKopf();
70         cout << "(1) aeussere Faktoren" << endl;
71         //Gibt den Spannungswert aus
72         cout << "Spannung [Volt]: " << meineFaktoren.getSpannung() << endl;
73         //Gibt die Temperatur aus
74         cout << "Temperatur [Grad Celsius]: " << meineFaktoren.getTemperatur() << endl;
75         //Gibt den Prozessfaktor aus
76         cout << "Prozessor (1=slow, 2=typical, 3=fast) : " << meineFaktoren.getProzess() << "\n\n" << endl;
77         cout << "(2) Bibliothek" << endl;
78         //Gibt Pfad zur Bibliotheksdatei aus
79         cout << "Pfad zur Bibliotheksdatei: "<< meineBibliothek.getPfad()<< "\n\n" << endl;
80         cout << "(3) Schaltwerk" << endl;
81         //Gibt Pfad zur Schaltnetzdatei aus
82         cout << "Pfad zur Schaltwerksdatei: " << meinSignalListeErzeuger.getPfadSchaltnetzdatei() << "\n\n" << endl;
83         cout << "(4) Analyse starten\n\n" << endl;
84         cout << "(5) Testwerte einfuegen\n\n" << endl;
85         cout << "(6) Programm beenden\n\n\n" << endl;
86         cout << "Waehle einen Menuepunkt und bestaetige mit Enter: ";
87
88         cin >> menuepunktstring;
89         menuepunkt = atoi(menuepunktstring.c_str());
90         //Falls keine Zahl zwischen 1 und 6 eingegeben wurde
91         if( cin.fail() || menuepunkt > 6 || menuepunkt < 1 || !checkEingabe(menuepunktstring)) {
92             //gebe Fehler aus
```

```
93              cout << "\nUngueltige Eingabe! \n\n";
94              menuepunkt = 0;
95              pause();
96          }
97          //Löscht Errorbits und Eingabgepuffer von cin
98          clear_cin();
99
100         switch( menuepunkt ) {
101             //Startet die Methode, welche das Untermenue Faktoren ↩
                    ausgibt
102             case 1:
103                 faktorenMenue();
104                 break;
105             //Startet die Methode, welche das Untermenue Bibliothek ↩
                    ausgibt
106             case 2:
107                 bibliothekMenue();
108                 break;
109             //Startet die Methode, welche das Untermenue Schaltwerk ↩
                    ausgibt
110             case 3:
111                 schaltwerkMenue();
112                 break;
113             //Startet die Analyse
114             case 4:
115                 analyse();
116                 break;
117             //Fügt Testwerte ein
118             case 5:
119                 testWerteEinfuegen();
120                 cout << "\nPfad zur Bibliotheksdatei erfolgreich ↩
                        eingetragen!\n";
121                 cout << "\n\nPfad zur Schaltnetzdatei erfolgreich ↩
                        eingetragen!\n\n";
122                 pause();
123                 break;
124             //Beendet das Programm
125             case 6:
126                 return;
127         }
128     }
129 }
130
131 /*
132  * Diese Methode gibt das Untermenue Faktoren auf dem Bildschirm aus.
133  * Durch Eingabe einer Zahl (1-5) kann man die Werte für die Spannung,↩
        Temperatur
134  * und den Prozessfaktor eintragen. Die berechneten Faktoren können ↩
       mit (4)
135  * ausgegeben werden und mit (5) gelangt man zurück ins Hauptmenü
136  */
137 void Menue::faktorenMenue() {
138     unsigned int menuepunkt_faktoren = 0;
139     string menuepunkt_faktorenstring = "";
140     //Läuft solange bis return kommt
141     while( true ) {
142         screenLoeschen();
143
144         menueKopf();
```

```
145         cout << "Untermenue Aeussere Faktoren" << endl;
146         //Spannungswert ausgeben
147         cout << "(1) Spannung [Volt]: " << meineFaktoren.getSpannung()
                << endl;
148         //Temperatur ausgeben
149         cout << "(2) Temperatur [Grad Celsius]: " << meineFaktoren.
               getTemperatur() << endl;          //Prozessfaktor ausgeben
150         cout << "(3) Prozess (1=slow, 2=typical, 3=fast): "<<
               meineFaktoren.getProzess() << endl;
151         cout << "(4) Ausgabe errechneter Faktoren" << endl;
152         cout << "(5) Hauptmenue\n\n\n" << endl;
153         cout << "Waehle einen Menuepunkt aus und bestaetige mit Enter:
                ";
154         cin >> menuepunkt_faktorenstring;
155         menuepunkt_faktoren = atoi(menuepunkt_faktorenstring.c_str());
156         cout << "\n\n-----------------------------------------" <<
               endl;
157
158         double faktor;
159         string faktorstring;
160
161         //Wenn kein int-Wert oder keine Zahl zwischen 1 und 6
               eingebeben wurde
162          if( cin.fail() || menuepunkt_faktoren > 5 ||
                menuepunkt_faktoren < 1 || !checkEingabe(
                menuepunkt_faktorenstring)) {
163             //gebe Fehler aus
164             cout << "\nUngueltige Eingabe!\n";
165             menuepunkt_faktoren = 0;
166             pause();
167          }
168          //Löscht Errorbits und Eingabgepuffer von cin
169          clear_cin();
170
171          switch( menuepunkt_faktoren ) {
172             case 1:
173                 //Spannungswert eingeben
174                 cout << "\nSpannung eingeben [Von 1.08 bis 1.32]: ";
175                 cin >> faktor;
176                 //Falls keine Zahl eingegeben wurde
177                 if( cin.fail() ) {
178                     //gebe Fehler aus
179                     cout << "Ungueltige Eingabe!";
180                 //ansonsten trage Spannung ein und setze zugehörige
                       bool-Variable auf true
181                 } else {
182                     meineFaktoren.setSpannung(faktor);
183                     spannung_gesetzt = true;
184                     pause();
185                 }
186                 clear_cin();
187                 break;
188             case 2:
189                 //Temperatur eingeben
190                 cout << "\nTemeratur eingeben [Von -25 bis 125]: ";
191                 cin >> faktor;
192                 //Falls keine Zahl eingegeben wurde
193                 if( cin.fail() ) {
194                     //gebe Fehler aus
```

```
195                     cout << "Ungueltige Eingabe";
196                 //ansonsten trage Temperatur ein und setze zugehörige ↩
                       bool-Varibale auf true
197                 } else {
198                     meineFaktoren.setTemperatur(faktor);
199                     temperatur_gesetzt = true;
200                     pause();
201                 }
202                 clear_cin();
203                 break;
204             case 3:
205                 //Prozessfaktor eingeben
206                 cout << "\nProzessfaktor eingeben: ";
207                 cin >> faktorstring;
208                  faktor = atoi(faktorstring.c_str());
209                 //Falls keine Zahl eingeben wurde
210                 if( cin.fail() || !checkEingabe(faktorstring)) {
211                     //gebe Fehler aus
212                     cout << "Ungueltige Eingabe";
213                     pause();
214                 //ansonsten trage Prozessfaktor ein und setze ↩
                       zugehörige bool-Varibale auf true
215                 } else {
216                     meineFaktoren.setProzess(faktor);
217                     bool prozess_gesetzt = true;
218                     pause();
219                 }
220                 clear_cin();
221                 break;
222             //Gibt alle berechneten Faktoren aus
223             case 4:
224                 meineFaktoren.ausgabeFaktoren();
225                 pause();
226                 break;
227             //return um die while-Schleife zu verlassen und ins ↩
                   Hauptmenü zurückzukehren
228             case 5:
229                  return;
230         }
231     }
232
233 }
234
235 /*
236  * Diese Methode gibt das Untermenue Bibliothek auf dem Bildschirm aus↩
        .
237  * Durch Eingabe einer Zahl (1-3) kann man den Pfad zur ↩
       Bibliotheksdatei eintragen,
238  * die Bibliotheksdatei ausgeben oder zurück ins Hauptmenü gelangen.
239  */
240 void Menue::bibliothekMenue() {
241     int menuepunkt_bibliothek = 0;
242     string menuepunkt_bibliothekstring;
243      //Läuft solange bis return kommt
244      while( true ) {
245         screenLoeschen();
246
247         menueKopf();
248         //Gibt den Pfad zur Bibliotheksdatei aus
```

```
249           cout << "(1) Pfad zur Bibliotheksdatei: " << meineBibliothek.↩
                 getPfad() << endl; //Pfad anzeigen
250           cout << "(2) Ausgabe der Bibliotheksdatei" << endl;
251           cout << "(3) Hauptmenue\n\n\n"  << endl;
252           cout << "Waehle einen Menuepunkt und bestaetige mit Enter: ";
253           cin >> menuepunkt_bibliothekstring;
254           cout << "----------------------------------------" << endl;
255           menuepunkt_bibliothek = atoi(menuepunkt_bibliothekstring.↩
                 c_str());
256           //Falls keine Zahl zwischen 1 und 3 eingegeben wurde
257           if( cin.fail() || menuepunkt_bibliothek > 3 || ↩
                 menuepunkt_bibliothek < 1 || !checkEingabe(↩
                 menuepunkt_bibliothekstring)) {
258              //gebe Fehler aus
259              cout << "\nUngueltige Eingabe!\n";
260               menuepunkt_bibliothek = 0;
261              pause();
262         }
263
264           clear_cin();
265           string pfad = "";
266
267           switch( menuepunkt_bibliothek ) {
268              //Pfad zur Bibliothek eingeben
269              case 1:
270                   cout << "Pfad zur Bibliotheksdatei eingeben: ";
271                   cin >> pfad;
272                   //Wenn der Pfad erfolgreich eingelesen werden konnte, ↩
                        eintragen des Pfades
273                   //und zugehörige bool-Variable auf true setze
274                   if( meineBibliothek.pfadEinlesen(pfad) ){
275                        meineBibliothek.dateiAuswerten();
276                        bibliothekspfad_gesetzt = true;
277                        cout << "Pfad erfolgreich eingetragen!\n";
278                   //ansonsten gebe Fehler aus
279                   } else {
280                        cout << "Pfad falsch oder Datei kann nicht ↩
                             geoeffnet werden!" << endl;
281                   }
282                   clear_cin();
283                   pause();
284                   break;
285              //Bibliotheksdatei ausgeben
286              case 2:
287                   meineBibliothek.dateiAusgabe();
288                   cout << "\n";
289                   pause();
290                   break;
291              //Ins Hauptmenü zurückkehren
292              case 3:
293                   return;
294            }
295
296      }
297  }
298
299 /*
300  * Diese Methode gibt das Untermenue Schaltwerk auf dem Bildschirm aus↩
       .
```

```
301  * Durch Eingabe einer Zahl (1-5) kann man den Pfad zur Schaltnetzdatei eintragen,
302  * die Schaltnetzdatei, die Signale oder die Graphstruktur ausgeben und ins
303  * Menü zurückkehren.
304  */
305 void Menue::schaltwerkMenue() {
306      int menuepunkt_schaltwerk = 0;
307     string menuepunkt_schaltwerkstring;
308      //Solange bis return kommt
309      while( true ) {
310          screenLoeschen();
311 
312          menueKopf();
313          cout << "Untermenue Schaltwerk" << endl;
314          //Gebe Pfad zur Schaltnetzdatei aus
315          cout << "(1) Pfad zur Schaltnetzdatei: " << meinSignalListeErzeuger.getPfadSchaltnetzdatei() << endl; //Pfad angeben
316          cout << "(2) Ausgabe der Schaltnetzdatei" << endl;
317          cout << "(3) Ausgabe der Signale" << endl;
318          cout << "(4) Ausgabe der Graphstruktur" << endl;
319          cout << "(5) Hauptmenue\n\n" << endl;
320          cout << "Waehle einen Menuepunkt und bestaetige mit Enter: ";
321          cin >> menuepunkt_schaltwerkstring;
322          cout << "----------------------------------------" << endl;
323          menuepunkt_schaltwerk = atoi(menuepunkt_schaltwerkstring.c_str());
324          //Falls keine Zahl zwischen 1 und 5 eingegeben wurde
325          if( cin.fail() || menuepunkt_schaltwerk > 5 || menuepunkt_schaltwerk < 1 || !checkEingabe(menuepunkt_schaltwerkstring)) {
326              //gebe Fehler aus
327              cout << "\nUngueltige Eingabe!\n";
328               menuepunkt_schaltwerk = 0;
329              pause();
330         }
331 
332         clear_cin();
333 
334         string pfad = "";
335 
336           switch( menuepunkt_schaltwerk ) {
337             //Pfad zur Schaltnetzdatei eingeben
338             case 1:
339                 cout << "Pfad zur Schaltnetzdatei eingeben: ";
340                 cin >> pfad;
341                 //Wenn eine existierende .txt Datei als Pfad angebeben wurde
342                 if( meinSignalListeErzeuger.setPfadSchaltnetzdatei(pfad) ) {
343                      signale = meinSignalListeErzeuger.erzeugeSignalliste();
344                   //falls es sicht bei der .txt-Datei nicht um eine Schaltwerksdatei handelt, wird der Pfad
345                   //zwar einegtragen, die bool-Variable bleibt jedoch false, also ist ein starten der
346                   //Analyse nicht möglich
347                   if( signale == NULL ) {
```

```
348                         cout << "Pfad nicht eingetragen!\n";
349                         cout << "Starten der Analyse nicht moeglich!\n"
                                " << endl;
350                         schaltnetzpfad_gesetzt = false;
351                     //trage den Pfad ein und setze die zugehörige bool
                        -Varibable auf true
352                     } else {
353                         cout << "Pfad erfolgreich eingetragen!\n";
354                         schaltnetzpfad_gesetzt = true;
355                     }
356                  //Gebe Fehler aus, dass der Pfad nicht korrekt ist
357                 } else {
358                     cout << "Pfad falsch oder Datei kann nicht
                        geoeffnet werden.\n" << endl;
359                 }
360                 clear_cin();
361                 pause();
362                 break;
363             //Gebe die Schaltnetzdatei aus
364             case 2:
365                 meinSignalListeErzeuger.ausgabeSchaltnetzdatei();
366                 pause();//Schaltnetzdatei ausgeben
367                 break;
368             //Gebe die Singale aus
369             case 3:
370                 meinSignalListeErzeuger.ausgabeSignale();
371                 pause();//Signal ausgeben
372                 break;
373             //Setzt im Grapherzeuger die zum Erzeugen benötigten
                Elemente
374             //und startet, wenn alles korrekt, ist die Erzeugung des
                Graphen und die Ausgabe
375             //auf dem Bildschirm
376             case 4:
377                 //Ausgabe der Graphstruktir nur möglich, falls der
                    Pfad zur Bibliothektsdatei gesetzt ist
378                 if( bibliothekspfad_gesetzt == true ) {
379                     meinGraphErzeuger.setBibliothek(&meineBibliothek);
380                     meinGraphErzeuger.setAnzahlSignale(
                            meinSignalListeErzeuger.getAnzahlSignale());
381                     meinGraphErzeuger.setSignale(signale);
382
383                     if( meinGraphErzeuger.erzeugeGraph() ) {
384                     meinGraphErzeuger.ausgabeGraph();
385                     }
386
387                  } else {
388                     cout << "\nPfad zur Bibliotheksdatei nicht
                        eingetragen!\n" << endl;
389                     cout << "Ausgabe der Graphstruktur nicht moeglich
                        !\n" << endl;
390                  }
391                 pause();
392                 break;
393             //Ins Hauptmenü zurückkehren
394             case 5:
395               return;
396         }
397
```

```
398     }
399   }
400
401 /*
402  * Diese Methode trägt für die Spannungswerte,Temperatur und den ↩
        Prozessfaktor Beispielwerte
403  * ein und setzt die Pfade für die Bibliotheksdatei, sowie die ↩
        Schaltnetzdatei.
404  * (Das setzen der Pfade ist nur möglich, falls sich die .txt Dateien ↩
        im Projektordner befinden)
405  *  Sie setzt ebenfalls alle bool-Variablen auf true, damit die ↩
        Analyse starten kann
406  */
407 void Menue::testWerteEinfuegen() {
408       double spannung = 1.2;
409       double temperatur = 55;
410       short prozess = 1;
411       string pfad_bib = "bib.txt";
412       string pfad_schaltnetz = "csd.txt";
413
414       //alle Faktoren eintragen
415       meineFaktoren.setSpannung(spannung);
416       meineFaktoren.setTemperatur(temperatur);
417       meineFaktoren.setProzess(prozess);
418
419       //alle Pfade eintragen
420       meineBibliothek.pfadEinlesen(pfad_bib);
421       meineBibliothek.dateiAuswerten();
422       meinSignalListeErzeuger.setPfadSchaltnetzdatei(pfad_schaltnetz);
423       signale = meinSignalListeErzeuger.erzeugeSignalliste();
424
425       //alle bool-Variablen auf true setzen
426       spannung_gesetzt = true;
427       temperatur_gesetzt = true;
428       prozess_gesetzt = true;
429       bibliothekspfad_gesetzt = true;
430       schaltnetzpfad_gesetzt = true;
431   }
432
433 //Diese Methode startet die Analyse
434 void Menue::analyse() {
435
436     //Starte die Analyse nur, falls zuvor alle benötigten Werte/Pfade↩
          eingetragen wurden
437     if ( !( spannung_gesetzt && temperatur_gesetzt && prozess_gesetzt
438               && bibliothekspfad_gesetzt && schaltnetzpfad_gesetzt )↩
                  ) {
439       //ansonsten gebe Fehler aus
440       cout << "\nLaufzeitanalyse fehlgeschlagen!\n" << endl;
441       cout << "\nErforderliche Werte oder Pfade fuer die Analyse ↩
            nicht eingetragen?\n" << endl;
442       cout << "Falls erforderliche Wert und Pfade eingetragen wurden↩
            : " << endl;
443       cout << "Angegebene Datei keine Schaltnetzdatei?\n\n" << endl;
444       pause();
445       return;
446      }
447       //Setze die für die Analyse benötigten Elemente
448       meinGraphErzeuger.setBibliothek(&meineBibliothek);
```

```
449         meinGraphErzeuger.setAnzahlSignale(meinSignalListeErzeuger.↩
                getAnzahlSignale());
450         meinGraphErzeuger.setSignale(signale);
451         meinGraphErzeuger.erzeugeGraph();
452         meinLaufzeitAnalysator.setFaktoren(&meineFaktoren);
453         meinLaufzeitAnalysator.setStartElement(meinGraphErzeuger.↩
                getStartElement());
454      //falls nicht alle Elemente korrekt sind
455      if( !meinLaufzeitAnalysator.start_LZA() ) {
456         //gebe Fehler aus
457         cout << endl << "LZA Fehlgeschlagen!" << endl;
458      }
459     //ansonsten gebe jeweils den längsten Pfad und die maximale ↩
           Laufzeit im
460     //Überführungs bzw. Ausgangsschaltnetz aus
461     else {
462         cout << "\nLaengster Pfad im Ueberfuehrungsschaltnetz:" << ↩
              endl; //ABFANGEN!!! Vergleich der berechneten mit der ↩
              Frequenz aus der SINGALLLISTE!!!
463         cout << meinLaufzeitAnalysator.getUebergangspfad() << endl;
464         cout << "Maximale Laufzeit der Pfade im ↩
              Ueberfuehrungsschaltnetz: ";
465         cout << meinLaufzeitAnalysator.getLaufzeitUebergangspfad()↩
              /1000 << " ps" << endl << endl;
466         //erzeuge 50 mal -
467         cout << string( 50, '-' ) << endl;
468         cout << "Laengster Pfad im Ausgangsschaltnetz:" << endl;
469         cout << meinLaufzeitAnalysator.getAusgangspfad() << endl;
470         cout << "Maximale Laufzeit der Pfade im Ausgangsschaltnetz: ";
471         cout << meinLaufzeitAnalysator.getLaufzeitAusgangspfad()/1000 ↩
              << " ps" << endl << endl;
472         cout << string( 50, '-' ) << endl;
473
474
475          //abfangen, ob keine FlipFlops im Netz vorhanden sind
476          if( !meinLaufzeitAnalysator.flipflopsVorhanden() ){
477                 cout << "Es sind keine FlipFlops im Schaltnetz ↩
                        vorhanden. " << endl;
478                 cout << "Schaltnetz ist unabhaengig von der Frequenz!"↩
                         << endl;
479                 pause();
480                 return;
481
482          } else{
483                 cout << "Maximale Frequenz: " << ↩
                        meinLaufzeitAnalysator.getFrequenz()/1e6 << " MHz\↩
                        n" << endl;
484          }
485
486
487         // Vergleicht die berechnete maximal zulässige Frequenz mit ↩
              der Taktfrequenz aus der Schaltnetzdatei
488         // Falls die Frequenz aus der Schaltnetzdatei größer als die ↩
              berechnete ist
489         if ( meinLaufzeitAnalysator.getFrequenz() < ↩
              meinSignalListeErzeuger.getFrequenz() ) {
490              //gebe Fehler aus
491              cout << "\nBedingung fuer die Taktfrequenz vom ↩
                     Schaltnetz/-werk ist nicht erfuellt!" << endl;
```

```
492                 cout << "Die Taktfrequenz " << meinSignalListeErzeuger↩
                        .getFrequenz()/1e6 <<" MHz ist groesser als die ↩
                        maximale Frequenz!\n" << endl;
493         } else {
494                 cout << "\n\n Bedingung fuer die Taktfrequenz vom ↩
                        Schaltnetz/-werk ist erfuellt!" << endl;
495         }
496     }
497 pause();
498 }
499
500 //leert und resetet den Eingabestream
501 void Menue::clear_cin(){
502     cin.clear();
503     cin.ignore(255, '\n');
504 }
505
506 bool Menue::checkEingabe(string str) {
507     if(str.length() > 1) {
508         return false;
509     }
510     return true;
511 }
512
513 //Gibt den Menuekopf aus
514 void Menue::menueKopf(){
515     cout << "****************************************" << endl;
516     cout << "*     IT-Projektpraktikum WS2011/2012    *" << endl;
517     cout << "* Laufzeitanalyse synchroner Schaltwerke *" << endl;
518     cout << "****************************************\n\n" << endl;
519 }
```

A.2 Klasse Bibliothek

Listing A.3: Inhalt der Datei Bibliothek.h

```
1  /*
2   * Datei:  Bibliothek.h
3   * Author: Maxim Köhler
4   *
5   * IT Praktikum WS 2012/13
6   * Gruppe 57
7   */
8
9  #ifndef BIBLIOTHEK_H
10 #define BIBLIOTHEK_H
11
12 #include <string>
13 #include <vector>
14 #include <fstream>
15 #include "GatterTyp.h"
16
17 using namespace std;
18
19 class Bibliothek {
20 private:
21     vector<GatterTyp*> bibElemente;
22     string datei;
23
```

```
24      void openError();
25      void readError();
26
27  public:
28      Bibliothek( string pfad = "" );
29      ~Bibliothek();
30      string getPfad();
31      GatterTyp* getBibElement( string typ );
32      void dateiAusgabe();
33      void dateiAuswerten();
34      bool pfadEinlesen( string pfad );
35
36  };
37
38  #endif  /* BIBLIOTHEK_H */
```

Listing A.4: Inhalt der Datei Bibliothek.cpp

```
1   /*
2    * Datei:  Bibliothek.cpp
3    * Author: Maxim Köhler
4    *
5    * IT Praktikum WS 2012/13
6    * Gruppe 57
7    */
8
9   #include "Bibliothek.h"
10  #include "Flipflop.h"
11  #include <istream>
12  #include <sstream>
13  using namespace std;
14
15
16  //Diese Methode gibt eine Fehlermeldung, dass die Datei nicht ↩
        geoeffnet werden konnte, aus
17  void Bibliothek::openError() {
18      cout << "FEHLER: Die Datei konnte nicht geoeffnet werden!" << endl↩
            ; //PAUSE!!!
19  }
20
21  //Diese Methode gibt eine Fehlermeldung, dass die Datei nicht gelesen ↩
        werden konnte, aus
22  void Bibliothek::readError() {
23      cout << "FEHLER beim Lesen der Datei!" << endl; //PAUSE!!!
24  }
25
26  /*
27   * Diese Methode gibt einen Zeiger auf den GatterTyp zurück, der den ↩
        Namen,
28   * den man über den String typ der Funktion übergibt, hat
29   */
30  GatterTyp* Bibliothek::getBibElement( string typ ) {
31      unsigned long length = bibElemente.size();
32
33      /*
34       * Für jeden GatterTyp im vector bibElemente wird überprüft, ob ↩
            der Name
35       * mit typ übereinstimmt, und gegebenenfalls zurückgegeben
36       */
```

```
37      for( int i = 0; i < length; i++ ) {
38          if( bibElemente.at( i )->getName() == typ )
39              return bibElemente.at( i );
40      }
41      cout << endl << "GatterTyp nicht gefunden!" << endl;
42      return NULL;
43  }
44
45  //gibt die Bibliothekdatei mit nummerierten Zeilen aus
46  void Bibliothek::dateiAusgabe() {
47      //Datei öffnen
48      ifstream file;
49      file.open( Bibliothek::datei.c_str(), ios::in );
50
51      //überprüfen, ob Datei geöffnet werden konnte
52      if ( !file.is_open() ) {
53          openError();
54          return;
55      }
56
57      string line;
58      int counter = 0;
59
60      //Für jede Zeile der Datei: Zeilennummer und die Zeile an sich ↩
            ausgeben
61      while( !file.eof()) {
62          //Überprüfung, ob Zeile gelesen werden kann
63          if ( getline( file, line ))
64              cout << " #" << counter << ":\t" << line << endl;
65          else
66              readError();
67          counter++;
68      }
69  }
70
71  //Wertet die Datei aus und speichert die gefundenen Gattertypen im ↩
        vector bibElemente
72  void Bibliothek::dateiAuswerten() {
73      //Datei öffnen
74      ifstream file;
75      file.open( Bibliothek::datei.c_str(), ios::in );
76
77      //Überprüfung, ob die Datei geöffnet werden konnten
78      if( !file.is_open() ) {
79          openError();
80          return;
81      }
82
83      string line;
84      GatterTyp* tempGatter = new GatterTyp();
85      Flipflop* tempFF = NULL;
86      bool currentIsFF = false;
87
88      //Liest jede einzelne Zeile in einer while-Schleife aus
89      while( !file.eof() ) {
90          //Kann Zeile gelesen werden?
91          if( getline( file, line )) {
92
93              //Handelt es sich um eine Zeile, die einen neuen GatterTyp↩
```

```
                beginnt?
94              if( line[0] == '[' && line[1] != '[' ) {
95
96                  //Der zuletzt bearbeitete Gattertyp wird gespeichert, falls er fehlerfrei ist, sonst gelöscht
97                  //Unterscheidung zwischen GatterTyp und Flipflop
98                  if( !currentIsFF && tempGatter->isValid() )
99                      bibElemente.push_back( tempGatter );
100                 else if( currentIsFF && tempFF->isValid() )
101                     bibElemente.push_back( tempFF );
102                 else
103                     delete tempGatter;
104
105                 //Ein neuer Gattertyp wird erstellt und der in der Zeile angegebene Name gespeichert
106                 //Das Flipflop dff wird gesondert behandelt
107                 if( line.find( "dff" ) == 1 ) {
108                     currentIsFF = true;
109                     tempFF = new Flipflop();
110                     tempFF->setName( "dff" );
111                     tempGatter = tempFF;
112                 }
113                 else {
114                     currentIsFF = false;
115                     tempGatter = new GatterTyp();
116                     tempGatter->setName( line.substr( 1, line.find( "] " )-1 ));
117                 }
118             }
119
120             //Spaghetticode, der alle möglichen Fälle für Zeilen abarbeitet und dementsprechend Attribute speichert
121             //Zeile gibt Eingaenge an?
122             else if( line.find( "ei" ) == 0 || line.find( "ed" ) == 0 || line.find( "et" ) == 0) {
123                 short ei;
124                 stringstream ss( line.substr( 3 ));
125                 if( ss >> ei ) {
126                     tempGatter->setEingaenge( ei );
127                 }
128             }
129             //Zeile gibt Grundlaufzeit an?
130             else if( line.find( "tpd0" ) == 0 || line.find( "tpdt" ) == 0 ) {
131                 double tpd0;
132                 stringstream ss( line.substr( 5 ));
133                 if( ss >> tpd0 ) {
134                     tempGatter->setGrundLaufzeit( tpd0 );
135                 }
136             }
137             //Zeile gibt Lastfaktor an?
138             else if( line.find( "kl" ) == 0 ) {
139                 short kl;
140                 stringstream ss( line.substr( 3 ));
141                 if( ss >> kl ) {
142                     tempGatter->setLastFaktor( kl );
143                 }
144             }
145             //Zeile gibt Lastkapazitaet an?
```

```
146             else if( line.find( "cl" ) == 0 || line.find( "cd" ) == 0 ↩
                    ) {
147                 short cl;
148                 stringstream ss( line.substr( 3 ));
149                 if( ss >> cl ) {
150                     tempGatter->setLastKapazitaet( cl );
151                 }
152             }
153             //Fängt die zusätzlichen Fälle für Flipflops ab
154             else if( currentIsFF ) {
155                 //Zeile gibt Setupzeit an?
156                 if( line.find( "tsetup" ) == 0 ) {
157                     short tsetup;
158                     stringstream ss( line.substr( 7 ));
159                     if( ss >> tsetup ) {
160                         tempFF->setSetupTime( tsetup );
161                     }
162                 }
163                 //Zeile gibt Haltezeit an?
164                 else if( line.find( "thold" ) == 0 ) {
165                     short thold;
166                     stringstream ss( line.substr( 6 ));
167                     if( ss >> thold ) {
168                         tempFF->setHoldTime( thold );
169                     }
170                 }
171                 //Zeile gibt Takt-Lastkapazität an?
172                 else if( line.find( "ct" ) == 0 ) {
173                     short ct;
174                     stringstream ss( line.substr( 3 ));
175                     if( ss >> ct ) {
176                         tempFF->setLastKapazitaetClock( ct );
177                     }
178                 }
179             }
180         }
181         else {
182             readError();
183             delete tempGatter;
184             return;
185         }
186     }
187 
188     //Speichern des zuletzt bearbeiteten Gattertyps, falls dieses ↩
        fehlerfrei ist
189     //Sonst: Speicherbereinigung
190     if( !currentIsFF && tempGatter->isValid() )
191         bibElemente.push_back( tempGatter );
192     else if( currentIsFF && tempFF->isValid() )
193         bibElemente.push_back( tempFF );
194     else
195         delete tempGatter;
196 }
197 
198 //Liest den Pfad zur Bibliotheksdatei ein
199 bool Bibliothek::pfadEinlesen( string pfad ) {
200     //Datei öffnen
201     ifstream file;
202     file.open( pfad.c_str(), ios::in );
```

```
203
204     //konnte Datei geöffnet werden?
205     if( file ) {
206         Bibliothek::datei = pfad;
207         file.close();
208         return true;
209     }
210     else {
211         return false;
212     }
213 }
214
215 //Konstruktor: Initialisierung des pfad-Attributs
216 Bibliothek::Bibliothek( string pfad ) {
217     if( pfad == "" )
218         return;
219
220     //Überprüfung, ob Datei vorhanden ist
221     ifstream file;
222     file.open( pfad.c_str(), ios::in );
223     if( file.is_open() ) {
224         datei = pfad;
225     }
226     else {
227         datei = "";
228     }
229 }
230
231 //Destruktor: Speicherbereinigung
232 Bibliothek::~Bibliothek() {
233     unsigned long length = bibElemente.size();
234
235     //gibt den Speicher aller bibElemente frei
236     for( int i = 0; i < length; i++ ) {
237         delete bibElemente.at( i );
238     }
239 }
240
241 //gibt den Pfad zur Bibliotheksdatei zurück
242 string Bibliothek::getPfad() {
243     return datei;
244 }
```

A.3 Klasse GatterTyp

Listing A.5: Inhalt der Datei `GatterTyp.h`

```
1  /*
2   * Datei:  GatterTyp.h
3   * Author: Maxim Köhler
4   *
5   * IT Praktikum WS 2012/13
6   * Gruppe 57
7   */
8
9  #ifndef GATTERTYP_H
10 #define GATTERTYP_H
11
12 #include <iostream>
```

```
13 using namespace std;
14
15 class GatterTyp {
16
17
18 protected:
19     string name;
20     double grundLaufzeit;
21     short lastFaktor;
22     short lastKapazitaet;
23     short eingaenge;
24
25 public:
26     GatterTyp();
27     ~GatterTyp();
28     string getName();
29     double getGrundLaufzeit();
30     short getLastFaktor();
31     short getLastKapazitaet();
32     short getEingaenge();
33     virtual bool getIsFlipflop();
34     void setName( string n );
35     void setGrundLaufzeit( double gl );
36     void setLastFaktor( short lf );
37     void setLastKapazitaet( short lk );
38     void setEingaenge( short ei );
39     bool isValid();
40     void out();
41
42
43 };
44
45 #endif  /* GATTERTYP_H */
```

Listing A.6: Inhalt der Datei `GatterTyp.cpp`

```
1  /*
2   * Datei:   GatterTyp.cpp
3   * Author:  Maxim Köhler
4   *
5   * IT Praktikum WS 2012/13
6   * Gruppe 57
7   */
8
9  #include "GatterTyp.h"
10
11 using namespace std;
12
13 //Konstruktor: Initialisiert die Attribute
14 GatterTyp::GatterTyp() {
15     name = "";
16     grundLaufzeit = -1.0;
17     lastFaktor = lastKapazitaet = eingaenge = -1;
18 }
19
20 //Destruktor
21 GatterTyp::~GatterTyp() {
22 }
23
```

```
24 //Überprüft, ob das Gatter fehlerfrei ist
25 bool GatterTyp::isValid() {
26     return ( name != "" && grundLaufzeit >= 0.0 && lastFaktor >= 0 && ↩
           lastKapazitaet >= 0 && eingaenge >= 0 );
27 }
28
29 //gibt zurück, ob Gatter ein Flipflop ist
30 bool GatterTyp::getIsFlipflop() {
31     return false;
32 }
33
34 //gibt den Namen zurück
35 string GatterTyp::getName() {
36     return name;
37 }
38
39 //gibt die Grundlaufzeit zurück
40 double GatterTyp::getGrundLaufzeit() {
41     return grundLaufzeit;
42 }
43
44 //gibt den Lastfaktor zurück
45 short GatterTyp::getLastFaktor() {
46     return lastFaktor;
47 }
48
49 //gibt die Lastkapazität zurück
50 short GatterTyp::getLastKapazitaet() {
51     return lastKapazitaet;
52 }
53
54 //Gibt die Eingänge zurück
55 short GatterTyp::getEingaenge() {
56     return eingaenge;
57 }
58
59 //setzt den Namen
60 void GatterTyp::setName( string n ) {
61     name = n;
62 }
63
64 //setzt die Grundlaufzeit
65 void GatterTyp::setGrundLaufzeit( double gl ) {
66     if( gl >= 0.0 )
67         grundLaufzeit = gl;
68 }
69
70 //setzt den Lastfaktor
71 void GatterTyp::setLastFaktor( short lf ) {
72     if( lf >= 0 )
73         lastFaktor = lf;
74 }
75
76 //setzt die Lastkapazitaet
77 void GatterTyp::setLastKapazitaet( short lk ) {
78     if( lk >= 0 )
79         lastKapazitaet = lk;
80 }
81
```

```
82 //setzt die Eingaenge
83 void GatterTyp::setEingaenge( short ei ) {
84     if( ei >= 0 )
85         eingaenge = ei;
86 }
87
88 //gibt alle Attribute aus
89 void GatterTyp::out () {
90     cout << "GatterTyp '" << name << "':" << endl;
91     cout << "Grundlaufzeit: " << grundLaufzeit << endl;
92     cout << "Lastfaktor: " << lastFaktor << endl;
93     cout << "Lastkapazitaet: " << lastKapazitaet << endl;
94     cout << "Eingaenge: " << eingaenge << endl << endl;
95 }
```

A.4 Klasse FlipFlop

Listing A.7: Inhalt der Datei FlipFlop.h

```
1  /*
2   * Datei:  Flipflop.h
3   * Author: Maxim Köhler
4   *
5   * IT Praktikum WS 2012/13
6   * Gruppe 57
7   */
8
9  #ifndef FLIPFLOP_H
10 #define FLIPFLOP_H
11
12 #include "GatterTyp.h"
13
14 using namespace std;
15
16 class Flipflop: public GatterTyp{
17 private:
18      short setupTime;
19      short holdTime;
20      short lastKapazitaetClock;
21
22 public:
23      Flipflop();
24      ~Flipflop();
25      bool getIsFlipflop();
26      short getSetupTime();
27      short getHoldTime();
28      short getLastKapazitaetClock();
29      void setSetupTime(short st);
30      void setHoldTime(short ht);
31      void setLastKapazitaetClock(short lkc);
32      bool isValid();
33      void out();
34 };
35
36
37 #endif  /* FLIPFLOP_H */
```

Listing A.8: Inhalt der Datei FlipFlop.cpp

```cpp
/*
 * Datei:   Flipflop.cpp
 * Author:  Maxim Köhler
 *
 * IT Praktikum WS 2012/13
 * Gruppe 57
 */

#include "Flipflop.h"
#include "GatterTyp.h"

//Konstrutor: Initialisiert die Attribute
Flipflop::Flipflop() {
    setupTime = holdTime = lastKapazitaetClock = -1;
}

//Destruktor
Flipflop::~Flipflop() {
}

//setzt die Setupzeit
void Flipflop::setSetupTime (short st) {
    if( st >= 0 )
        setupTime = st;
}

//setzt die Haltezeit
void Flipflop::setHoldTime(short ht) {
    if( ht >= 0 )
        holdTime = ht;
}

//setzt die Takt-Lastkapazität
void Flipflop::setLastKapazitaetClock(short lkc) {
    if( lkc >= 0 )
        lastKapazitaetClock = lkc;
}

//gibt die Setupzeit zurück
short Flipflop::getSetupTime() {
    return setupTime;
}

//gibt die Haltezeit zurück
short Flipflop::getHoldTime() {
    return holdTime;
}

//gibt die Takt-Lastkapazität zurück
short Flipflop::getLastKapazitaetClock() {
    return lastKapazitaetClock;
}

//gibt zurück, dass es sich um ein Flipflop handelt
bool Flipflop::getIsFlipflop() {
    return true;
}

//gibt alle Attribute aus
```

```
60 void Flipflop::out() {
61     cout << "Flipflop '" << name << "':" << endl;
62     cout << "Grundlaufzeit: " << grundLaufzeit << endl;
63     cout << "Lastfaktor: " << lastFaktor << endl;
64     cout << "Lastkapazitaet: " << lastKapazitaet << endl;
65     cout << "Eingaenge: " << eingaenge << endl;
66     cout << "Setupzeit: " << setupTime << endl;
67     cout << "Haltezeit: " << holdTime << endl;
68     cout << "Takt-Lastkapazitaet: " << lastKapazitaetClock;
69 }
70
71 //gibt zurück, ob das Objekt fehlerfrei ist
72 bool Flipflop::isValid() {
73     return (name != "" && grundLaufzeit >= 0.0 && lastFaktor >= 0 && ↩
           lastKapazitaet >= 0
74          && eingaenge >= 0 && setupTime >= 0 && holdTime >= 0 && ↩
              lastKapazitaetClock >= 0);
75 }
```

A.5 Klasse Faktoren

Listing A.9: Inhalt der Datei Faktoren.h

```
1  /*
2   * Datei:  Faktoren.h
3   * Author: Sebastian Müller
4   *
5   * IT Praktikum WS 2012/13
6   * Gruppe 57
7   */
8
9  #ifndef FAKTOREN_H
10 #define FAKTOREN_H
11
12 using namespace std;
13
14 class Faktoren {
15
16 private:
17     double spannung;
18     double temperatur;
19     short prozess;
20     double spannungFaktor;
21     double temperaturFaktor;
22     double prozessFaktor;
23
24     bool berechneSpannungFaktor();
25     bool berechneTemperaturFaktor();
26     bool berechneProzessFaktor();
27     double berechneFaktor(double wert, double arr[][2], int laenge);
28     double interpolation(double wert, double x1, double x2, double y1,↩
           double y2);
29
30 public:
31     Faktoren();
32     ~Faktoren();
33
34     double getSpannung() const;
35     double getTemperatur() const;
```

```
36      short getProzess() const;
37      void getFaktoren(double& spgFaktor, double& tmpFaktor, double& ↩
            przFaktor) const;
38      bool setSpannung(double spannung);
39      bool setTemperatur(double temperatur);
40      bool setProzess(short prozess);
41      void ausgabeFaktoren();
42
43  };
44
45  #endif  /* FAKTOREN_H */
```

Listing A.10: Inhalt der Datei Faktoren.cpp

```
1   /*
2    * Datei:  Faktoren.cpp
3    * Author: Sebastian Müller
4    *
5    * IT Praktikum WS 2012/13
6    * Gruppe 57
7    */
8
9   #include <iostream>
10  #include <sstream>
11  #include "Faktoren.h"
12  using namespace std;
13
14  //Konstruktor: initalisiert alle Größen mit Wert 0
15  Faktoren::Faktoren() {
16      Faktoren::spannung = 0;
17      Faktoren::temperatur = 0;
18      Faktoren::prozess = 0;
19      Faktoren::spannungFaktor = 0;
20      Faktoren::temperaturFaktor = 0;
21      Faktoren::prozessFaktor = 0;
22  }
23
24  //Tut nichts.
25  Faktoren::~Faktoren() { }
26
27  //Findet gemittelten Wert im R^2
28  double Faktoren::interpolation(double wert, double x1, double x2, ↩
        double y1, double y2) {
29      //Steigung per Steigungsdreieck
30      double steigung = ( y2 - y1 ) / ( x2 - x1 );
31      //Anwenden des Steigungsdreiecks ausgehend von x1
32      double ergebnis = y1 + ( wert - x1 ) * steigung;
33
34      return ergebnis;
35  }
36
37  //Berechnet Faktor
38  double Faktoren::berechneFaktor(double wert, double arr[][2], int ↩
        laenge) {
39      //Lineare Suche zum finden des benötigten Wertes
40          for (int i = 0; i<laenge; i++) {
41              //Fall: Wert ist in Tabelle
42              if(arr[i][0] == wert) {
43                  return arr[i][1];
```

```
44                  break;
45              }
46              //Fall: Wert ist nicht in Tabelle
47              else if(arr[i][0] > wert) {
48                  return Faktoren::interpolation(wert, arr[i-1][0], arr[↩
                        i][0], arr[i-1][1], arr[i][1]);
49                  break;
50              }
51          }
52          return 0;
53  }
54
55  //Berechnet den Faktor der Spannung
56  bool Faktoren::berechneSpannungFaktor() {
57      //Prüfen ob Spannung in angegebenen Grenzen liegt
58      if(Faktoren::spannung >= 1.08 && Faktoren::spannung <= 1.32) {
59          //Wertetabelle aus Spezifikation
60          double werte[7][2] = { {1.08, 1.121557}, {1.12, 1.075332}, ↩
                {1.16, 1.035161}, {1.20, 1.000000}, {1.24, 0.968480}, ↩
                {1.28, 0.940065}, {1.32, 0.914482}};
61
62          //Berechnen
63          Faktoren::spannungFaktor = Faktoren::berechneFaktor(Faktoren::↩
                spannung, werte, 7);
64          cout << "\nSpannung erfolgreich eingetragen!\n\n";
65          return true;
66      }
67      //Spannung liegt nicht in angegebenen Grenzen
68      else {
69          cout << endl << "Spannung hat keinen gueltigen Wert!" << endl;
70          cout << "Bitte Spannung erneut eingeben!\n"<< endl;
71          return false;
72      }
73  }
74
75
76  bool Faktoren::berechneTemperaturFaktor() {
77      //Prüfen ob Temperatur in angegebenen Grenzen liegt
78      if(Faktoren::temperatur >= -25 && Faktoren::temperatur <= 125) {
79          //Wertetabelle
80          double werte[15][2] = { {-25, 0.897498}, {-15, 0.917532}, {0, ↩
                0.948338}, {15, 0.979213}, {25, 1.000000}, {35, 1.020305},↩
                 {45, 1.040540}, {55, 1.061831}, {65, 1.082983}, {75, ↩
                1.103817} ,{85, 1.124124}, {95, 1.144245}, {105, ↩
                1.164563}, {115, 1.184370}, {125, 1.204966} };
81
82          //Berechnen
83          Faktoren::temperaturFaktor = Faktoren::berechneFaktor(Faktoren↩
                ::temperatur, werte, 15);
84          cout << "\nTemperatur erfolgreich eingetragen!\n\n";
85          return true;
86      }
87      else {
88          cout << endl << "Temperatur hat keinen gueltigen Wert!" << ↩
                endl;
89          cout << "Bitte Temperatur erneut eingeben!\n"<< endl;
90          return false;
91      }
92  }
```

```
93
94 //Berechnet Faktor für den Prozess
95 bool Faktoren::berechneProzessFaktor() {
96     //Einfache Abfrage der 3 möglichen Werte, ansonsten Fehlermeldung
97     if (Faktoren::prozess == 1) {
98         Faktoren::prozessFaktor = 1.174235;
99         cout << "\nProzessfaktor erfolgreich eingetragen!\n\n";
100        return true;
101    }
102    else if (Faktoren::prozess == 2) {
103        Faktoren::prozessFaktor = 1.000000;
104        cout << "\nProzessfaktor erfolgreich eingetragen!\n\n";
105        return true;
106    }
107    else if (Faktoren::prozess == 3) {
108        Faktoren::prozessFaktor = 0.876148;
109        cout << "\nProzessfaktor erfolgreich eingetragen!\n\n";
110        return true;
111    }
112    //Fehler:
113    else {
114        cout << endl << "Prozess hat keinen gueltigen Wert (1, 2, 3)!"↩
               << endl;
115        cout << "Bitte Prozessfaktor erneut eingeben!\n"<< endl;
116        return false;
117    }
118
119}
120
121double Faktoren::getSpannung() const {
122    return Faktoren::spannung;
123}
124
125double Faktoren::getTemperatur() const {
126    return Faktoren::temperatur;
127}
128
129short Faktoren::getProzess() const {
130    return Faktoren::prozess;
131}
132
133void Faktoren::getFaktoren(double& spgFaktor, double& tmpFaktor, ↩
   double& przFaktor) const {
134    spgFaktor = Faktoren::spannungFaktor;
135    tmpFaktor = Faktoren::temperaturFaktor;
136    przFaktor = Faktoren::prozessFaktor;
137}
138
139bool Faktoren::setSpannung(double spannung) {
140    double temp = Faktoren::spannung;
141    Faktoren::spannung = spannung;
142    //Faktor berechnen und auf Fehler überprüfen
143    if(Faktoren::berechneSpannungFaktor()) {
144        return true;
145    }
146    else {
147        Faktoren::spannung = temp;
148        return false;
149    }
```

```
150 }
151
152 bool Faktoren::setTemperatur(double temperatur) {
153     double temp = Faktoren::temperatur;
154     Faktoren::temperatur = temperatur;
155     //Faktor berechnen und auf Fehler überprüfen
156     if(Faktoren::berechneTemperaturFaktor()) {
157         return true;
158     }
159     else {
160         Faktoren::temperatur = temp;
161         return false;
162     }
163 }
164
165 bool Faktoren::setProzess(short prozess) {
166     double temp = Faktoren::prozess;
167     Faktoren::prozess = prozess;
168     //Faktor berechnen und auf Fehler überprüfen
169     if(Faktoren::berechneProzessFaktor()) {
170         return true;
171     }
172     else {
173         Faktoren::prozess = temp;
174         return false;
175     }
176 }
177
178 void Faktoren::ausgabeFaktoren() {
179     //Konvertieren der Double in String-Werte
180     stringstream NumberString;
181     NumberString << Faktoren::spannungFaktor;
182     string KV = NumberString.str();
183     NumberString.str("");
184     NumberString << Faktoren::temperaturFaktor;
185     string KT = NumberString.str();
186     NumberString.str("");
187     NumberString << Faktoren::prozessFaktor;
188     string KP = NumberString.str();
189
190     //Ausgabe
191     cout << "\nFaktoren: KV: " + KV + " | KT: " + KT + " | KP: " + KP ↩
        + "\n" << endl;
192 }
```

A.6 Klasse Signal

Listing A.11: Inhalt der Datei `Signal.h`

```
1  /*
2   * Datei:  Signal.h
3   * Author: Kristian Maier
4   *
5   * IT Praktikum WS 2012/13
6   * Gruppe 57
7   */
8
9  #ifndef SIGNAL_H
10 #define SIGNAL_H
```

```
11
12 #include <string>
13
14 using namespace std;
15
16
17 class Signal {
18
19 public:
20     enum signalTypen {
21         EINGANG,
22         INTERN,
23         AUSGANG,
24         UNBEKANNT
25     };
26
27 private:
28
29     signalTypen signalTyp;
30     string quelle;
31     string quellenTyp;
32     string ziele[5];
33     int anzahlZiele;
34
35
36 public:
37
38     Signal();
39     ~Signal(){};
40
41     int getAnzahlZiele() const ;
42     signalTypen getSignalTyp() const;
43     string getQuelle() const;
44     string getQuellenTyp() const;
45     string getZiel(int pos) const;
46
47     void setAnzahlZiele(int anzahlZiele);
48     void setSignalTyp(signalTypen sigTyp);
49     void setQuelle(string quelle);
50     void setQuellenTyp(string quellenTyp);
51     void zielHinzufuegen(string gatterName, int pos);
52
53 };
54
55 #endif  /* SIGNAL_H */
```

Listing A.12: Inhalt der Datei `Signal.cpp`

```
1 /*
2  * Datei:  Signal.cpp
3  * Author: Kristian Maier
4  *
5  * IT Praktikum WS 2012/13
6  * Gruppe 57
7  */
8
9 #include "Signal.h"
10
11
```

```
12 Signal::Signal(){
13     signalTyp = UNBEKANNT;
14     anzahlZiele=0;
15     quelle = "";
16     quellenTyp = "";
17     for(int i=0; i<5; i++){
18         ziele[i] = "";
19     }
20 }
21
22 int Signal::getAnzahlZiele() const {
23     return anzahlZiele;
24 }
25
26 Signal::signalTypen Signal::getSignalTyp() const{
27     return Signal::signalTyp;
28 }
29
30 string Signal::getQuelle() const {
31     return quelle;
32 }
33
34 string Signal::getQuellenTyp() const {
35     return quellenTyp;
36 }
37
38 string Signal::getZiel(int pos) const{
39     //Test ob Arraygrenzen eingehalten werden
40     if(pos>=0 && pos <5){
41         return ziele[pos];
42     }
43     return "";
44 }
45
46 void Signal::setAnzahlZiele(int anzahlZiele) {
47     this->anzahlZiele = anzahlZiele;
48 }
49
50 void Signal::setSignalTyp(signalTypen sigTyp){
51     signalTyp = sigTyp;
52 }
53
54 void Signal::setQuelle(string quelle) {
55     this->quelle = quelle;
56 }
57
58 void Signal::setQuellenTyp(string quellenTyp) {
59     this->quellenTyp = quellenTyp;
60 }
61
62 void Signal::zielHinzufuegen(string gatterName, int pos){
63     //Test ob Arraygrenzen eingehalten werden
64     if(pos>=0 && pos <5){
65         ziele[pos] = gatterName;
66         anzahlZiele++;
67     }
68
69 }
```

A.7 Klasse SignalListeErzeuger

Listing A.13: Inhalt der Datei SignalListeErzeuger.h

```
1  /*
2   * Datei:  SignalListeErzeuger.h
3   * Author: Kristian Maier
4   *
5   * IT Praktikum WS 2012/13
6   * Gruppe 57
7   */
8
9  #ifndef SIGNALLISTEERZEUGER_H
10 #define SIGNALLISTEERZEUGER_H
11
12
13 #include <fstream>
14 #include <string>
15 #include <iostream>
16 #include <sstream>
17
18 #include "Signal.h"
19
20 using namespace std;
21
22
23 class SignalListeErzeuger {
24 private:
25     Signal* signale;
26     short anzahlSignale;
27     string datei;
28     long frequenz;
29
30     void ausgabeOpenError();
31     void ausgabeReadError();
32
33
34     //Hilfsfunktionen für Erzeugung der Schaltnetzdatei
35     void readEntity(string& input,string& output,string&
36                         intern,string& clock);
37     int  countSignals(string& input,string& output,string& intern);
38     void fillList(string input, string output, string intern);
39     long readFrequenz(string& clock);
40     int extractSignal(string& str);
41     bool readGatterInfo();
42     bool updateSignallisteMitGatter(string& line);
43
44     //Hilfsfunktionen für Stringoperationen
45     string readUntilChar(string& str, char chr) const;
46     void eraseUntilChar(string& str, char chr);
47     void eraseAfterChar(string& str, char chr);
48
49 public:
50     bool setPfadSchaltnetzdatei(string pfad);
51     void setFrequenz(long frequenz);
52
53     short getAnzahlSignale() const;
54     string getPfadSchaltnetzdatei() const;
55     long getFrequenz() const;
```

```
56
57      Signal* erzeugeSignalliste();
58      void ausgabeSchaltnetzdatei();
59      void ausgabeSignale();
60
61
62
63  };
64
65  #endif  /* SIGNALLISTEERZEUGER_H */
```

Listing A.14: Inhalt der Datei SignalListeErzeuger.cpp

```
1   /*
2    * Datei:  SignalListeErzeuger.cpp
3    * Author: Kristian Maier
4    *
5    * IT Praktikum WS 2012/13
6    * Gruppe 57
7    */
8
9   #include "SignalListeErzeuger.h"
10
11  using namespace std;
12
13  void SignalListeErzeuger::setFrequenz(long frequenz){
14      this->frequenz = frequenz;
15  }
16
17
18  short SignalListeErzeuger::getAnzahlSignale() const{
19      return this->anzahlSignale;
20  }
21
22
23  string SignalListeErzeuger::getPfadSchaltnetzdatei() const{
24      return this->datei;
25  }
26
27
28  long SignalListeErzeuger::getFrequenz() const{
29      return this->frequenz;
30  }
31
32
33  /*
34   * Gibt den String von Anfang von str bis zum ersten Auftreten von chr↩
       zurueck.
35   * Gibt es kein chr, wird alles zurueckgegeben
36   */
37  string SignalListeErzeuger::readUntilChar(string& str, char chr) const↩
     {
38      stringstream ss(str);
39      string tmp;
40      getline(ss, tmp, chr);
41      return tmp;
42  }
43
44
```

```
45 /*
46  * Loescht innerhalb von str alles bis zum ersten Auftreten von chr
47  * inklusive chr. Gibt es dies nicht, wird alles geloescht.
48  *
49  */
50 void SignalListeErzeuger::eraseUntilChar(string& str, char chr){
51     stringstream ss(str);
52     string tmp;
53     getline(ss, tmp, chr);
54     str.erase(0, tmp.length() + 1);
55 }
56
57
58 /*
59  * Loescht im string str alles hinter dem ersten vorkommenden Zeichen
60  * chr inklusive dem Zeichen.
61  *
62  */
63 void SignalListeErzeuger::eraseAfterChar(string& str, char chr){
64     stringstream ss(str);
65     getline(ss, str, chr);
66 }
67
68
69 /*
70  * Öffnet die Datei wenn dies moeglich ist, sonst wird der Pfad
71  * nicht gespeichert. Der Rueckgabetyp gibt an, ob das oeffnen der ↩
        Datei
72  * erfolgreich war.
73  */
74 bool SignalListeErzeuger::setPfadSchaltnetzdatei(string pfad){
75     ifstream file (pfad.c_str(), ios::in);
76     if(file){
77         datei = pfad;
78         file.close();
79         return true;
80     }
81     return false;
82
83 }
84
85 /*
86  * Erzeugt die Signalliste und gibt sie als Zeiger auf ein dynamisches↩
        Array
87  * zurueck. Dazu wird die Schaltnetzdatei geoeffnet und ausgelesen. ↩
        Ist
88  * dies nicht moeglich, wird der Nullzeiger zurückgegeben und die ↩
        Listen-
89  * laenge betraegt den Wert 0. Eine entsprechende Fehlermeldung wird ↩
        ausgegeben.
90  *
91  * Die Laenge der Liste kann über getAnzahlSignale() abgerufen werden.
92  */
93 Signal* SignalListeErzeuger::erzeugeSignalliste(){
94
95     if(signale != NULL){
96        delete[] signale;
97        signale = NULL;
98     }
```

```
99      anzahlSignale = 0;
100
101
102     //Exceptions abfangen, falls die Schaltnetzdatei nicht ausgelesen ↩
            werden
103     //kann, weil sie z.B. fehlerhaft ist.
104     try{
105         string input, output, intern, clock;
106         //die Signalaufzaehlung auslesen
107         readEntity(input, output, intern, clock);
108         //Signale zaehlen
109         anzahlSignale = countSignals(input, output, intern);
110         //dynamisches Array erzeugen
111         signale = new Signal[anzahlSignale];
112         //Frequenz auslesen
113         frequenz = readFrequenz(clock);
114         //Signalliste erzeugen und den Signaltyp zuweisen
115         fillList(input, output, intern);
116
117         //Gatter bestimmen, welche durch die Signale verknuepft
118         //sind und Kurzschluss auswerten
119         if(readGatterInfo()){
120             cout << "Kurzschluss im Schaltnetz entdeckt" << endl;
121             return NULL;
122         }
123
124     }catch(...){
125        //Ausgabe einer Fehlermeldung bei Fehler
126         anzahlSignale = 0;
127         ausgabeReadError();
128         if(signale != NULL){
129             signale = NULL;
130             delete[] signale;
131         }
132         return NULL;
133     }
134
135     return signale;
136 }
137
138 /*
139  * Liest den Entity Teil der Schaltnetzdatei und speichert die Signale
140  * in den entsprechenden Referenzen.
141  */
142 void SignalListeErzeuger::readEntity(string& input, string& output,
143                                      string& intern, string& clock){
144
145
146     ifstream file (datei.c_str(), ios::in);
147
148     //auf Fehler beim oeffnen testen
149     if(!file){
150         ausgabeOpenError();
151         return;
152     }
153
154     string line;
155     int count = 0;
156     while(getline(file, line)){
```

```
157         if(line.find("INPUT") == 0){
158                 input = line;
159                 //"INPUT" am Anfang entfernen
160                 eraseUntilChar(input, ' ');
161                 //Semikolon am Ende entfernen
162                 eraseAfterChar(input, ';');
163                 count++;
164         }else if(line.find("OUTPUT") == 0){
165                 output = line;
166                 eraseUntilChar(output, ' ');
167                 eraseAfterChar(output, ';');
168                 count++;
169         }else if(line.find("SIGNALS") == 0){
170                 intern = line;
171                 eraseUntilChar(intern, ' ');
172                 eraseAfterChar(intern, ';');
173                 count++;
174         }else if(line.find("CLOCK") == 0){
175                 clock = line;
176                 eraseUntilChar(clock, ' ');
177                 eraseAfterChar(clock, ';');
178                 count++;
179          }
180         if(count == 4){
181             file.close();
182             return;
183         }
184     }
185     file.close();
186     //Fehler ausloesen, falls ENTITY Kopf nicht ausgelesen werden ↩
           konnte
187     throw 1;
188 }
189
190
191 /*
192  * Liefert die Anzahl der Signale zurueck, die in input, output, und ↩
       intern
193  * definiert sind, indem alle Zeichen zusammengezaehlt werden und ↩
       durch die
194  * Anzahl der Zeichen (=5) geteilt wird,  die ein Signal zur ↩
       Beschreibung
195  * braucht.
196  *
197  */
198 int SignalListeErzeuger::countSignals(string& input,string& output,↩
    string& intern){
199     return (input.length() + output.length() + intern.length() + 3)/5;
200
201 }
202
203 /*
204  * Weißt ihnen den Signalen den richtigen Signal Typ zu, aber noch ↩
       nicht Ziele
205  * und Quelle. Die Position im Array entspricht dabei der Signalnummer↩
        und kann
206  * so weiter eindeutig zugeordnet bleiben.
207  */
208 void SignalListeErzeuger::fillList(string input, string output, string↩
```

```
    intern){
209
210     //Alle zu erzeugende Signale durchgehen
211     for(int i = 0; i < anzahlSignale; i++){
212        //Den Typ des Signals ermitteln, indem der Index mit der Liste,↩
               die durch
213        //input, output und intern gegeben ist, verglichen wird. Die ↩
              Signale mit
214        //der niedrigsten Nummer stehen immer vorne in der Liste.
215        if(i+1 == extractSignal(input)){
216            signale[i].setSignalTyp(Signal::EINGANG);
217            //gespeichertes Signal aus Liste loeschen
218            eraseUntilChar(input, ',');
219        }
220        else if(i+1 == extractSignal(output)){
221            signale[i].setSignalTyp(Signal::AUSGANG);
222            //gespeichertes Signal aus Liste loeschen
223            eraseUntilChar(output, ',');
224
225        }
226        else if(i+1 == extractSignal(intern)){
227            signale[i].setSignalTyp(Signal::INTERN);
228            //gespeichertes Signal aus Liste loeschen
229            eraseUntilChar(intern, ',');    //gespeichertes Signal aus ↩
                  Liste loeschen
230
231        }
232     }
233
234 }
235
236
237 /*
238  * Liest den Teil zwischen BEGIN und END aus, um die Gatter zu ↩
       bestimmen
239  * welche durch die Signale verknuepft sind. Gibt außerdem true ↩
       zurueck,
240  * wenn Kurzschluesse erkannt wurden, sonst false;
241  */
242 bool SignalListeErzeuger::readGatterInfo(){
243
244     ifstream file (datei.c_str(), ios::in);
245
246     //auf Fehler beim oeffnen testen
247     if(!file){
248         return false;
249     }
250
251     string line;
252     //nach Finden von "BEGIN" in der Datei wird gatterInfo = true und ↩
          die
253     //folgenden Zeilen sind die Signalbeziehungen zu den Gattern bis "↩
          END"
254     bool gatterInfo = false;
255     while(getline(file, line)){
256         if(!gatterInfo  && (line.find("BEGIN") == 0)){
257             gatterInfo=true;
258             continue;
259         }
```

```
260         if(gatterInfo  && (line.find("END") == 0)){
261             break;
262         }
263         if(gatterInfo){
264             //Eine Zeile verarbeiten
265             //wenn Rueckgabe true ist, wurde ein Kurzschluss gefunden
266             if(updateSignallisteMitGatter(line)){
267                 return true;
268             }
269 
270         }
271     }
272     file.close();
273     return false;
274 }
275 
276 /*
277  * Liest die Informationen aus einer Zeile des Teils zwischen BEGIN ↩
        und END
278  * und speichert diese innerhalb der Signalliste ab. Erkennt außerdem ↩
        Kurz-
279  * schluesse im Schaltwerk und gibt true zurueck, wenn einer gefunden ↩
        wurde.
280  */
281 bool SignalListeErzeuger::updateSignallisteMitGatter(string& line){
282     string gatterName;
283     //Gattername steht am Anfang
284     gatterName = readUntilChar(line, ':');      //line=g003:dff(s024,↩
        clk,s043);
285     //Nach Auslesen loeschen
286     eraseUntilChar(line, ':');                  //line=dff(s024,clk,↩
        s043);
287 
288     string gatterTyp;
289     //Gattertyp steht jetzt am Anfang
290     gatterTyp = readUntilChar(line, '(');
291     eraseUntilChar(line, '(');                  //line=s024,clk,s043);
292 
293     line = readUntilChar(line, ')');            //line = s024,clk,s043
294     //hinteres Komma raussuchen, rechts davon steht das Ausgangssignal↩
            des Gatters
295     unsigned long letztesKomma = line.find_last_of(',', string::npos);
296     string ausgangssignal = line.substr(letztesKomma+1, string::npos);
297     //int wert des Signals bestimmen
298     int intAusgangssignal = extractSignal(ausgangssignal);
299 
300     //Referenz auf Ausgangssignal besorgen
301     Signal& sig = signale[intAusgangssignal-1];
302 
303     if((sig.getQuelle() != "") || (sig.getSignalTyp() == Signal::↩
        EINGANG)){
304         //Kurzschluss wenn Signal schon Quelle hat oder
305         //externer Eingang auf interner Ausgang
306         return true;
307     }
308     sig.setQuelle(gatterName);
309     sig.setQuellenTyp(gatterTyp);
310 
311     line = line.substr(0,letztesKomma);         //line = s024,clk
```

```
312
313     //Mehrere verbleibende Eingangssignale nacheinander einschreiben
314     while(!line.empty()){
315         string eingangssignal;
316         eingangssignal = readUntilChar(line,',');
317         eraseUntilChar(line, ',');                //line = clk
318         if(eingangssignal == "clk"){
319             //clk kann ignoriert werden
320            continue;
321         }
322         int intEingangssignal = extractSignal(eingangssignal);
323
324         //Referenz auf Eingangssignal besorgen
325         Signal& sig = signale[intEingangssignal-1];
326         //Ziel Gatter im Signal speichern
327         sig.zielHinzufuegen(gatterName, sig.getAnzahlZiele());
328     }
329
330     return false;
331
332 }
333
334 /*
335  * Im uebergebenen String befinden sich die durch Komma getrennten ↩
        Signale.
336  * Das erste Signal wird dabei in ein int Wert gewandelt.
337  */
338 int SignalListeErzeuger::extractSignal(string& str){
339     string number;
340     number = readUntilChar(str, ',');
341     number.erase(0,1);
342     stringstream ss(number);
343     int num;
344     ss >> num;
345     return num;
346 }
347
348 /*
349  * Gibt die Frequenz zurueck, die in der CLOCK Zeile angegeben war und
350  * mit clock uebergeben wurde.
351  */
352 long SignalListeErzeuger::readFrequenz(string& clock){
353     stringstream ss(clock);
354     string tmp;
355     long freq;
356
357     //clock hat jetzt das zB. Format "clk, ZZZZ MHz"
358
359     //clk, abschneiden:
360     getline(ss, tmp, ',');       //tmp="clk"
361
362     ss >> freq;                  //freq = ZZZZ
363     ss >> tmp;                   //tmp = MHz
364
365     if(tmp.compare("MHz") == 0){
366         freq*=1000000L;
367     }else if(tmp.compare("kHz") == 0){
368         freq*=1000L;
369     }//sonst in Hz
```

```
370     return freq;
371 }
372
373 /*
374  * Gibt die komplette Schaltnetzdatei auf der Konsole aus. Dazu wird die
375  * Schaltnetzdatei geoeffnet und ausgelesen. Ist dies nicht moeglich,
376  * wird eine entsprechende Fehlermeldung ausgegeben.
377  */
378 void SignalListeErzeuger::ausgabeSchaltnetzdatei(){
379     ifstream file (datei.c_str(), ios::in);
380
381     if(!file){
382         ausgabeOpenError();
383         return;
384     }
385
386     int i=1;
387     string line;
388     //Datei zeilenweise ausgeben mit Zeilennummern
389     while(getline(file, line)){
390         cout << "#" << i << ": " << line << endl;
391         i++;
392     }
393
394     file.close();
395
396 }
397
398 /*
399  * Gibt die Signalliste auf der Konsole aus. Dazu muss vorher die Liste
400  * ueber erzeugeSignalliste() erfolgreich erzeugt werden.
401  */
402 void SignalListeErzeuger::ausgabeSignale(){
403
404     if(anzahlSignale == 0){
405         cout << "Keine Signale vorhanden" << endl;
406         return;
407     }
408
409     cout << "Signale:" << endl;
410
411     for(int i=0; i < anzahlSignale; i++){
412         //Ausgabe Trennstriche
413         cout << "-----" << endl;
414
415         //Ausgabe Signalname
416         cout << "Signalname : s";
417         if(i+1 < 10){
418             cout << "00";
419         }else if(i+1 < 100){
420             cout << "0";
421         }
422         cout << i+1 << endl;
423
424         //Ausgabe Signaltyp
425         cout << "Signaltyp : ";
426         switch(signale[i].getSignalTyp()){
```

```
427             case Signal::AUSGANG:
428                 cout << "Ausgangssignal";
429                 break;
430             case Signal::EINGANG:
431                 cout << "Eingangssignal";
432                 break;
433             case Signal::INTERN:
434                 cout << "Internes Signal";
435                 break;
436             default:
437                 cout << "unbekanntes Signal";
438                 break;
439         }
440         cout << endl;
441
442
443         //Ausgabe der Signalquelle wenn vorhanden
444         cout << "Signalquelle : ";
445         if(signale[i].getSignalTyp() == Signal::EINGANG){
446             cout << "keine Quelle" << endl;
447         }else{
448             cout << signale[i].getQuelle() << endl;
449         }
450
451
452         int ziele = signale[i].getAnzahlZiele();
453         cout << "--> Das Signal hat " << ziele;
454         cout << " Ziele" << endl;
455
456         if(ziele > 0){
457             cout << "Ziel-Gatter :";
458             while(ziele){
459                 cout << " " << signale[i].getZiel(ziele-1);
460                 ziele--;
461             }
462             cout << endl;
463         }
464
465
466     }
467 }
468
469 /*
470  * Gibt eine Fehlermeldung aus, wenn die Schaltnetzdatei nicht
471  * geoeffnet werden konnte;
472  */
473 void SignalListeErzeuger::ausgabeOpenError(){
474     cout << "Fehler beim Oeffnen der Schaltnetzdatei"  << endl;
475     cout << "Pfad richtig oder Datei schon geoeffnet?" << endl;
476 }
477
478 /*
479  * Gibt eine Fehlermeldung aus, wenn die Schaltnetzdatei nicht
480  * ausgelesen werden konnte;
481  */
482 void SignalListeErzeuger::ausgabeReadError(){
483     cout << "Fehler beim Lesen der Schaltnetzdatei"  << endl;
484     cout << "Datei korrekt?" << endl;
485 }
```

A.8 Klasse ListenElement

Listing A.15: Inhalt der Datei ListenElement.h

```
1  /*
2   * Datei:  ListenElement.h
3   * Author: Sebastian Müller
4   *
5   * IT Praktikum WS 2012/13
6   * Gruppe 57
7   */
8
9  #ifndef LISTENELEMENT_H
10 #define LISTENELEMENT_H
11
12 #include "SchaltwerkElement.h"
13
14 using namespace std;
15
16 class ListenElement {
17 private:
18     SchaltwerkElement* schaltwerkElement;
19     ListenElement* next;
20 public:
21     ListenElement();
22     ~ListenElement();
23     SchaltwerkElement* getSchaltwerkElement();
24     ListenElement* getNextElement();
25     void setSchaltwerkElement(SchaltwerkElement* schaltwerkEl);
26     void setNextElement(ListenElement* nextEl);
27 };
28
29 #endif  /* LISTENELEMENT_H */
```

Listing A.16: Inhalt der Datei ListenElement.cpp

```
1  /*
2   * Datei:  ListenElement.cpp
3   * Author: Sebastian Müller
4   *
5   * IT Praktikum WS 2012/13
6   * Gruppe 57
7   */
8
9  #include "ListenElement.h"
10 #include "SchaltwerkElement.h"
11 #include <iostream>
12 using namespace std;
13
14
15
16 //Tut nichts.
17 ListenElement::~ListenElement() { }
18
19 //Getter
20 SchaltwerkElement* ListenElement::getSchaltwerkElement() {
21     return ListenElement::schaltwerkElement;
22 }
23
```

```
24 ListenElement* ListenElement::getNextElement() {
25     return ListenElement::next;
26 }
27
28 void ListenElement::setSchaltwerkElement(SchaltwerkElement* ↩
       schaltwerkEl) {
29     ListenElement::schaltwerkElement = schaltwerkEl;
30 }
31
32 void ListenElement::setNextElement(ListenElement* nextEl) {
33     ListenElement::next = nextEl;
34 }
35
36 //Setzt alle Zeiger auf NULL
37 ListenElement::ListenElement() {
38     ListenElement::setSchaltwerkElement(NULL);
39     ListenElement::setNextElement(NULL);
40 }
```

A.9 Klasse SchaltwerkElement

Listing A.17: Inhalt der Datei SchaltwerkElement.h

```
1  /*
2   * Datei:  SchaltwerkElement.h
3   * Author: Kristian Maier
4   *
5   * IT Praktikum WS 2012/13
6   * Gruppe 57
7   */
8
9  #ifndef SCHALTWERKELEMENT_H
10 #define SCHALTWERKELEMENT_H
11
12 #include <string>
13 #include "GatterTyp.h"
14
15
16 using namespace std;
17
18 class SchaltwerkElement {
19 private:
20
21     string name;
22     GatterTyp* typ;
23     double laufzeitEinzelgatter;
24     SchaltwerkElement* nachfolgerElemente[5];
25     int anzahlNachfolger;
26     bool isEingangsElement;
27     bool isAusgangsElement;
28     short anzahlEingangssignale;
29
30 public:
31     SchaltwerkElement(GatterTyp* gTyp);
32     ~SchaltwerkElement();
33
34
35     int getAnzahlNachfolger() const;
36     short getAnzahlEingangssignale() const;
```

```
37      bool getIsEingangsElement() const;
38      bool getIsAusgangsElement() const;
39      string getName() const;
40      double getLaufzeitEinzelgatter() const;
41
42      GatterTyp* getTyp() const;
43      SchaltwerkElement* getNachfolger(int pos) const;
44
45      void setName(string n);
46      void setAnzahlNachfolger(int anzahlN);
47      void setAnzahlEingangssignale(short anzahlE);
48      void setIsEingangsElement(bool isEingangsEl);
49      void setIsAusgangsElement(bool isAusgangsEl);
50      void setLaufzeitEinzelgatter(double lfz);
51
52      void nachfolgerHinzufuegen(SchaltwerkElement* schaltwerkElement, ↩
           int pos);
53
54
55  };
56
57  #endif  /* SCHALTWERKELEMENT_H */
```

Listing A.18: Inhalt der Datei `SchaltwerkElement.cpp`

```
1   /*
2    * Datei:  SchaltwerkElement.cpp
3    * Author: Kristian Maier
4    *
5    * IT Praktikum WS 2012/13
6    * Gruppe 57
7    */
8
9   #include <vector>
10  #include "SchaltwerkElement.h"
11
12  SchaltwerkElement::SchaltwerkElement(GatterTyp* gTyp){
13       this->typ = gTyp;
14
15       this->name = "NULL";
16       this->laufzeitEinzelgatter = 0;
17       this->anzahlNachfolger = 0;
18       this->anzahlEingangssignale = 0;
19   }
20
21  SchaltwerkElement::~SchaltwerkElement(){}
22
23  int SchaltwerkElement::getAnzahlNachfolger() const {
24      return anzahlNachfolger;
25  }
26  short SchaltwerkElement::getAnzahlEingangssignale() const {
27      return anzahlEingangssignale;
28  }
29
30
31  bool SchaltwerkElement::getIsAusgangsElement() const {
32      return isAusgangsElement;
33  }
34
```

```
35 bool SchaltwerkElement::getIsEingangsElement() const {
36     return isEingangsElement;
37 }
38
39 double SchaltwerkElement::getLaufzeitEinzelgatter() const {
40     return laufzeitEinzelgatter;
41 }
42
43 string SchaltwerkElement::getName() const {
44     return name;
45 }
46
47 GatterTyp* SchaltwerkElement::getTyp() const{
48     return typ;
49 }
50
51 SchaltwerkElement* SchaltwerkElement::getNachfolger(int pos) const{
52     if(pos < 0 || pos >= anzahlNachfolger){
53         return NULL;
54     }
55     return nachfolgerElemente[pos];
56 }
57
58 void SchaltwerkElement::setName(string n){
59     this->name = n;
60 }
61
62 void SchaltwerkElement::setAnzahlNachfolger(int anzahlN){
63     this->anzahlNachfolger = anzahlN;
64 }
65
66 void SchaltwerkElement::setAnzahlEingangssignale(short anzahlE){
67     this->anzahlEingangssignale = anzahlE;
68 }
69
70 void SchaltwerkElement::setIsEingangsElement(bool isEingangsEl){
71     this->isEingangsElement = isEingangsEl;
72 }
73
74 void SchaltwerkElement::setIsAusgangsElement(bool isAusgangsEl){
75     this->isAusgangsElement = isAusgangsEl;
76 }
77
78 void SchaltwerkElement::setLaufzeitEinzelgatter(double lfz){
79     this->laufzeitEinzelgatter = lfz;
80 }
81
82 void SchaltwerkElement::nachfolgerHinzufuegen(SchaltwerkElement* ↩
    schaltwerkElement, int pos){
83
84     if(!(pos < 0 || pos >= 5)){
85         nachfolgerElemente[pos] = schaltwerkElement;
86         anzahlNachfolger++;
87     }
88 }
```

A.10 Klasse GraphErzeuger

Listing A.19: Inhalt der Datei GraphErzeuger.h

```
1  /*
2   * Datei:  GraphErzeuger.h
3   * Author: Sebastian Müller
4   *
5   * IT Praktikum WS 2012/13
6   * Gruppe 57
7   */
8
9  #ifndef GRAPHERZEUGER_H
10 #define GRAPHERZEUGER_H
11
12 #include "Bibliothek.h"
13 #include "ListenElement.h"
14 #include "Signal.h"
15
16 class GraphErzeuger {
17 private:
18     Bibliothek* bibliothek;
19     ListenElement* startElement;
20     ListenElement* endElement;
21     Signal* signale;
22     short anzahlSignale;
23
24     bool erzeugeListe();
25     void push_back(ListenElement* elem);
26     SchaltwerkElement* sucheGatter(string gatter);
27     SchaltwerkElement* sucheNachNameInListe(string name);
28     bool signalBenutztTest(Signal* sig);
29     bool gatterFehlerTest(SchaltwerkElement* elem);
30
31
32 public:
33     GraphErzeuger();
34
35     //Kris Methoden
36     bool erzeugeGraph();
37     void ausgabeGraph();
38
39     //Sebastian Methoden
40     bool erzeugeGraph2();
41
42
43     void setBibliothek(Bibliothek* bib);
44     Bibliothek* getBibliothek();
45     void setStartElement(ListenElement* element);
46     ListenElement* getStartElement();
47     void setEndElement(ListenElement* element);
48     ListenElement* getEndElement();
49     void setSignale(Signal* sig);
50     inline Signal* getSignale();
51     void setAnzahlSignale(short signale);
52     short getAnzahlSignale();
53
54 };
55
56
57 #endif  /* GRAPHERZEUGER_H */
```

Listing A.20: Inhalt der Datei GraphErzeuger.cpp

```
1  /*
2   * Datei:  GraphErzeuger.cpp
3   * Author: Sebastian Müller
4   *
5   * IT Praktikum WS 2012/13
6   * Gruppe 57
7   */
8
9  #include "GraphErzeuger.h"
10 #include "SignalListeErzeuger.h"
11 using namespace std;
12
13 //Konstruktor: initalisiert Zeiger mit NULL
14 GraphErzeuger::GraphErzeuger() {
15     GraphErzeuger::startElement = NULL;
16     GraphErzeuger::endElement = NULL;
17     GraphErzeuger::signale = NULL;
18     GraphErzeuger::bibliothek = NULL;
19 }
20
21 //Standart Getter und Setter:
22 Bibliothek* GraphErzeuger::getBibliothek() {
23     return GraphErzeuger::bibliothek;
24 }
25
26 void GraphErzeuger::setBibliothek(Bibliothek* bib) {
27     GraphErzeuger::bibliothek = bib;
28 }
29
30 ListenElement* GraphErzeuger::getStartElement() {
31     return GraphErzeuger::startElement;
32 }
33
34 void GraphErzeuger::setStartElement(ListenElement* element) {
35     GraphErzeuger::startElement = element;
36 }
37
38 ListenElement* GraphErzeuger::getEndElement() {
39     return GraphErzeuger::endElement;
40 }
41
42 void GraphErzeuger::setEndElement(ListenElement* element) {
43     GraphErzeuger::endElement = element;
44 }
45
46 short GraphErzeuger::getAnzahlSignale() {
47     return GraphErzeuger::anzahlSignale;
48 }
49
50 void GraphErzeuger::setAnzahlSignale(short signale) {
51     GraphErzeuger::anzahlSignale = signale;
52 }
53
54
55 void GraphErzeuger::setSignale(Signal* sig){
56     this->signale = sig;
57 }
```

```
58
59 inline Signal* GraphErzeuger::getSignale() {
60     return GraphErzeuger::signale;
61 }
62
63 //Neues Element in Liste einfügen
64 void GraphErzeuger::push_back(ListenElement* elem) {
65     //Fall: kein Element in der Liste -> startPointer auf Element
66     if(GraphErzeuger::startElement == NULL) {
67         GraphErzeuger::startElement = elem;
68     }
69     //Fall: bereits Element in der Liste -> next-Pointer auf nächstes ↩
           Element richten
70     if(GraphErzeuger::endElement != NULL) {
71         GraphErzeuger::endElement->setNextElement(elem);
72     }
73     //endPointer auf neues Element setzen
74     GraphErzeuger::endElement = elem;
75
76     //Nächstes Element mit NULL-Pointer initalisieren
77     elem->setNextElement(NULL);
78 }
79
80 //Suche Gatter anhand des Namens
81 SchaltwerkElement* GraphErzeuger::sucheGatter(string gatter) {
82
83     //Geht gesamte Liste des Graphen durch
84     for(ListenElement * elem = GraphErzeuger::startElement; elem != ↩
           NULL; elem = elem->getNextElement()) {
85
86         //Fall: Zugehöriges Schaltwerkelement heißt wie gesuchtes
87         if(elem->getSchaltwerkElement()->getName() == gatter) {
88
89             //Gebe Schaltwerkelement-Pointer zurück
90             return elem->getSchaltwerkElement();
91         }
92     }
93     //Fall: nichts gefunden
94     cout << "Fehler! Gatter nicht in Liste!" << endl;
95     return NULL;
96 }
97
98 //Testet, ob ein Signal in ein Gatter führt, wenn es nicht als Ausgang↩
       deklariert wurde
99 bool GraphErzeuger::signalBenutztTest(Signal* sig) {
100
101     //Eingang: Muss Ziel haben
102     if(sig->getSignalTyp() == Signal::EINGANG && sig->getAnzahlZiele()↩
           < 1) {
103         return false;
104     }
105
106     //Internes Signal: muss Quelle und Ziel haben
107     else if(sig->getSignalTyp() == Signal::INTERN && (sig->↩
           getAnzahlZiele() < 1 || sig->getQuelle() == "")) {
108         return false;
109     }
110
111     //Ausgang: Muss Quelle haben
```

```
112     else if(sig->getSignalTyp() == Signal::AUSGANG && sig->getQuelle()↩
            == "") {
113         return false;
114     }
115 
116     return true;
117 }
118 
119 //Testet, ob die Anzahl der Eingangssignale mit der Bibliothek ↩
    übereinstimmt
120 bool GraphErzeuger::gatterFehlerTest(SchaltwerkElement* elem) {
121     if(elem->getTyp()->getEingaenge() == elem->↩
          getAnzahlEingangssignale()) {
122         return true;
123     }
124     else {
125         return false;
126     }
127 }
128 
129 //Erstellt die einfach verkettete Liste mit den bereits bekannten ↩
    Infos
130 bool GraphErzeuger::erzeugeListe() {
131 
132 //    Marker für Augsangssignal setzten.
133 
134     bool thisElemAusg = false;
135 
136     //Liste neu initalisieren
137     GraphErzeuger::setStartElement(NULL);
138     GraphErzeuger::setEndElement(NULL);
139 
140     //Gehe Signalliste durch
141     for(int i = 0; i < GraphErzeuger::anzahlSignale; i++) {
142 
143 
144         if(signale[i].getQuelle() != "") {
145 
146             //Fall Ausgangssignal: Marker setzen
147             if(signale[i].getSignalTyp() == Signal::AUSGANG) {
148                 thisElemAusg = true;
149             }
150 
151             //Listenelement erstellen
152             ListenElement* elem = new ListenElement;
153 
154             GatterTyp* gTyp = bibliothek->getBibElement(signale[i].↩
                getQuellenTyp());
155 
156             //SchaltwerkElement mit ListenElement verknüpfen und ↩
                bekannte Infos eintragen
157             elem->setSchaltwerkElement(new SchaltwerkElement(gTyp));
158             elem->getSchaltwerkElement()->setName(signale[i].getQuelle↩
                ());
159 
160             //Eingangselement ERSTMAL false setzten (wird später ↩
                nochmal überprüft)
161             elem->getSchaltwerkElement()->setIsEingangsElement(false);
162             elem->getSchaltwerkElement()->setIsAusgangsElement(↩
```

```
                thisElemAusg);
163
164             //ListenElement in Liste einfügen
165             GraphErzeuger::push_back(elem);
166
167             //Ausgangsmarker zurücksetzen
168             thisElemAusg = false;
169         }
170     }
171
172     //Eingangsgatter kann erst gesetzt werden, wenn die Liste bereits ↩
            existiert =(
173     for(int i = 0; i < GraphErzeuger::anzahlSignale; i++){
174         //Fall Eingangssignal:
175         if(signale[i].getSignalTyp() == Signal::EINGANG) {
176
177             //Alle Ziele als Eingangselement
178             for(int j = 0; j < signale[i].getAnzahlZiele(); j++) {
179                 sucheGatter((signale[i].getZiel(j)))->↩
                        setIsEingangsElement(true);
180             }
181         }
182     }
183     return true;
184 }
185
186 //Erstellt den Graphen
187 bool GraphErzeuger::erzeugeGraph() {
188
189     //Fehler abfangen
190     if(GraphErzeuger::bibliothek == NULL) {
191         cout << "Graph: Bibliothek nicht eingebunden!" << endl;
192         return false;
193     }
194     if(GraphErzeuger::signale == NULL) {
195         cout << "Graph: Signalliste nicht eingebunden!" << endl;
196         return false;
197     }
198
199     //Liste erzeugen (s.o.)
200     if(!erzeugeListe()) {
201         return false;
202     }
203
204     //Gehe alle Signale durch
205     for(int i = 0; i<GraphErzeuger::anzahlSignale; i++) {
206
207         //Alle Ziele der Signale durchgehen
208         for(int j = 0; j < signale[i].getAnzahlZiele(); j++) {
209
210             //Gibt es ein Ziel, so erhöhe die Anzahl der Eingänge des ↩
                    Ziels um 1
211             if(signale[i].getZiel(j) != "") {
212                 short signaleAlt = sucheGatter(signale[i].getZiel(j))↩
                        ->getAnzahlEingangssignale();
213                 sucheGatter(signale[i].getZiel(j))->↩
                        setAnzahlEingangssignale(signaleAlt + 1);
214
215                 //Gibt es eine Quelle, so füge alle Ziele als ↩
```

```
                    Nachfolger der Quelle hinzu
216                 if(signale[i].getQuelle() != "") {
217                     //Jedes Ziel als Nachfolger hinzufügen
218                     sucheGatter(signale[i].getQuelle())->↩
                            nachfolgerHinzufuegen(sucheGatter(signale[i].↩
                            getZiel(j)), j);
219                 }
220             }
221
222         }
223
224     }
225
226     //Test auf Unbenutzte Signale und falsch beschaltetete Gatter
227
228     //Signalliste erneut durchgehen
229     for(int i = 0; i<GraphErzeuger::anzahlSignale; i++) {
230
231         //s.o.
232         if(!signalBenutztTest(&signale[i])) {
233             cout << endl << "Fehler!";
234             cout << endl << "Unbenutztes Signal!" << endl;
235             return false;
236         }
237
238         //Quellgatter des Signals überprüfen, falls existent (s.o.)
239         if(signale[i].getQuelle() != "" && !gatterFehlerTest(↩
                sucheGatter(signale[i].getQuelle()))) {
240             cout << endl << "Fehler!";
241             cout << endl << sucheGatter(signale[i].getQuelle())->↩
                    getName() << " hat zu viele / zu wenige ↩
                    Eingangssignale!" << endl;
242             return false;
243         }
244     }
245     return true;
246 }
247
248 //Ausgabefunktion für die Konsole
249 void GraphErzeuger::ausgabeGraph(){
250     cout << endl;
251
252     //Gehe Liste durch und gebe die Infos aus
253     for(ListenElement* elem = getStartElement(); elem != NULL; elem = ↩
            elem->getNextElement()) {
254         cout << "Gatterame: " << elem->getSchaltwerkElement()->getName↩
                () << endl;
255         cout << "Gattertyp: " << elem->getSchaltwerkElement()->getTyp↩
                ()->getName() << endl;
256         cout << "-->Das Gatter hat " << elem->getSchaltwerkElement()->↩
                getAnzahlNachfolger() << " Ziele" << endl;
257         cout << "Angeschlossene Gatter : ";
258         for(int i = 0; i < elem->getSchaltwerkElement()->↩
                getAnzahlNachfolger(); i++) {
259             cout << elem->getSchaltwerkElement()->getNachfolger(i)->↩
                    getName() << " ";
260         }
261         cout << endl << "--------------------------------" << endl;
262     }
```

```
263
264 }
```

A.11 Klasse LaufzeitAnalysator

Listing A.21: Inhalt der Datei LaufzeigAnalysator.h

```
1  /*
2   * Datei:  LaufzeitAnalysator.h
3   * Author: Kristian Maier
4   *
5   * IT Praktikum WS 2012/13
6   * Gruppe 57
7   */
8
9  #ifndef LAUFZEITANALYSATOR_H
10 #define LAUFZEITANALYSATOR_H
11
12 #include <map>
13 #include <iostream>
14 #include <string>
15 #include "ListenElement.h"
16 #include "Faktoren.h"
17 #include "Flipflop.h"
18
19 using namespace std;
20
21 class LaufzeitAnalysator {
22 private:
23     Faktoren* faktoren;
24     ListenElement* startElement;
25     long frequenz;
26     string uebergangspfad;
27     string ausgangspfad;
28     double laufzeitUebergangspfad;
29     double laufzeitAusgangspfad;
30
31     struct DFS_Daten {
32         SchaltwerkElement* vaterElement;
33         double pfadLaufzeit;
34     };
35     map<SchaltwerkElement*, DFS_Daten> DFS_Zwischenspeicher;
36
37
38     void berechneLaufzeitEinzelgatter();
39     bool dfs(ListenElement* start);
40     bool dfs_visit(SchaltwerkElement* k, SchaltwerkElement* s, string ↩
           pfad);
41     double berechneFrequenz(SchaltwerkElement* flipFlop, double ↩
           laufzeitUeberfuehrung);
42     bool zyklensuche(SchaltwerkElement* v);
43
44 public:
45     LaufzeitAnalysator();
46
47     void setFaktoren(Faktoren* fak);
48     void setStartElement(ListenElement* start);
49
50     bool start_LZA();
```

```
51      bool flipflopsVorhanden();
52
53      long getFrequenz();
54      string getUebergangspfad();
55      string getAusgangspfad();
56      double getLaufzeitUebergangspfad();
57      double getLaufzeitAusgangspfad();
58
59  };
60
61  #endif  /* LAUFZEITANALYSATOR_H */
```

Listing A.22: Inhalt der Datei LaufzeitAnalysator.cpp

```
1
2   /*
3    * Datei:  LaufzeitAnalysator.cpp
4    * Author: Kristian Maier
5    *
6    * IT Praktikum WS 2012/13
7    * Gruppe 57
8    */
9
10  #include "LaufzeitAnalysator.h"
11
12  using namespace std;
13
14
15  LaufzeitAnalysator::LaufzeitAnalysator(){
16      faktoren = NULL;
17      startElement = NULL;
18      frequenz = 0;
19      uebergangspfad = "";
20      ausgangspfad = "";
21      laufzeitUebergangspfad = 0;
22      laufzeitAusgangspfad = 0;
23  }
24
25  void LaufzeitAnalysator::setFaktoren(Faktoren* fak){
26      this->faktoren = fak;
27  }
28
29  void LaufzeitAnalysator::setStartElement(ListenElement* start){
30      this->startElement = start;
31  }
32
33
34  long LaufzeitAnalysator::getFrequenz(){
35      return this->frequenz;
36  }
37
38  string LaufzeitAnalysator::getUebergangspfad(){
39      return this->uebergangspfad;
40  }
41
42  string LaufzeitAnalysator::getAusgangspfad(){
43      return this->ausgangspfad;
44  }
45
```

```
46 double LaufzeitAnalysator::getLaufzeitUebergangspfad(){
47     return this->laufzeitUebergangspfad;
48 }
49
50 double LaufzeitAnalysator::getLaufzeitAusgangspfad(){
51     return this->laufzeitAusgangspfad;
52 }
53
54 /*
55  * Startet die Laufzeitanalyse des Schaltwerks. Dazu muss vorher das Start-
56  * element und die Faktoren gesetzt sein.
57  */
58 bool LaufzeitAnalysator::start_LZA(){
59     //eventuell alte Berechnungen zuruecksetzen
60     uebergangspfad = "";
61     ausgangspfad = "";
62     laufzeitUebergangspfad = 0;
63     laufzeitAusgangspfad = 0;
64     frequenz = 0;
65
66     berechneLaufzeitEinzelgatter();
67
68     //fuer alle gueltigen Startknoten(Eingangsgatter oder FF) die Tiefensuche aufrufen
69     SchaltwerkElement* sE;
70     for(ListenElement* k = startElement; k->getNextElement() != NULL; k = k->getNextElement()){
71         sE = k->getSchaltwerkElement();
72         if(sE->getIsEingangsElement() || sE->getTyp()->getIsFlipflop()){
73             if(!dfs(k)) { //Tiefensuche aufrufen
74                 return false;
75             }
76         }
77     }
78     return true;
79 }
80
81
82 /*
83  * Tiefensuche vorbereiten und durchfuehren
84  */
85 bool LaufzeitAnalysator::dfs(ListenElement* start){
86
87     //alte Analysen verwerfen
88     DFS_Zwischenspeicher.clear();
89
90     //alle Knoten = SchaltwerkElemente in der Map initialisieren mit 0
91     SchaltwerkElement* sE;
92     for(ListenElement* k = this->startElement; k->getNextElement() != NULL; k = k->getNextElement()){
93         sE = k->getSchaltwerkElement();
94         DFS_Zwischenspeicher[sE].pfadLaufzeit = 0;
95         DFS_Zwischenspeicher[sE].vaterElement = NULL;
96     }
97
98     sE = start->getSchaltwerkElement();
99     //Tiefensuche durchfuehren
```

```
100     if(!dfs_visit(sE,sE, "")) {
101         return false;
102     }
103 
104     return true;
105 }
106 
107 
108 //Zyklensuche: Überprüft, ob die Folgeknoten des übergebenen Knoten ↩
    bereits entdeckt wurden
109 bool LaufzeitAnalysator::zyklensuche(SchaltwerkElement* v) {
110 
111 //     Vorherige Version:
112 //     if(LaufzeitAnalysator::DFS_Zwischenspeicher[v].pfadLaufzeit > 0)↩
    {
113 //         return true;
114 //     }
115 
116     //Fall: Betrachteter Knoten hat überhaupt Nachfolger
117     if(v->getAnzahlNachfolger() > 0) {
118 
119         //Gehe alle Nachfolger durch...
120         for(int i = 0; i < v->getAnzahlNachfolger(); i++) {
121 
122             //...und Prüfe, ob schon eine Pfadlaufzeit für diese ↩
                gesetzt wurde (=schon entdeckt)
123             if(LaufzeitAnalysator::DFS_Zwischenspeicher[v->↩
                getNachfolger(i)].pfadLaufzeit > 0) {
124                 cout << endl << "Fehler! Zyklus an " << v->↩
                    getNachfolger(i)->getName() << " gefunden!" << ↩
                    endl;
125                 return true;
126             }
127             //return zyklensuche(v->getNachfolger(i));
128         }
129     }
130     return false;
131 }
132 
133 
134 /*
135  * Fuehrt die Tiefensuche durch, verwendet den vorgegeben Pseudo-Code
136  */
137 bool LaufzeitAnalysator::dfs_visit(SchaltwerkElement* k, ↩
    SchaltwerkElement* s, string pfad){
138     SchaltwerkElement* v;
139     double tempZeit;
140 
141     //verfolgter Pfad speichern
142     pfad += "->" + k->getName();
143 
144     //alle Nachfolger iterieren
145     for(int i = 0; i < k->getAnzahlNachfolger(); i++){
146         v = k->getNachfolger(i);
147         tempZeit = DFS_Zwischenspeicher[k].pfadLaufzeit + k->↩
            getLaufzeitEinzelgatter();
148 
149         //wenn Flipflop ist Endelement des Uebergangpfades gefunden
150         if(v->getTyp()->getIsFlipflop()){
```

```
151             //wenn neue Laufzeit groesser, neue Maximale Laufzeit ↩
                    setzen
152             if(laufzeitUebergangspfad < tempZeit){
153                 laufzeitUebergangspfad = tempZeit;
154                 uebergangspfad = pfad + "->" + v->getName();     //↩
                        Pfadstring der groessten Laufzeit speichern
155                 Flipflop* ff = (Flipflop*)(v->getTyp());         //↩
                        Referenz auf den FlipFloptyp
156                 //berechne Frequenz in Hz
157                 frequenz = (long) ( 1e15 / ( laufzeitUebergangspfad + ↩
                        1000 * ff->getSetupTime()) );
158             }
159         //sonst wenn moeglicher laengerer Pfad
160         }else if(DFS_Zwischenspeicher[v].pfadLaufzeit < tempZeit){
161             //möglicher Zyklus pruefen
162             if(((DFS_Zwischenspeicher[v].pfadLaufzeit != 0) || (v == s↩
                    )) &&
163                  DFS_Zwischenspeicher[v].vaterElement != k){
164 
165                  DFS_Zwischenspeicher[v].vaterElement = k;
166                  if(zyklensuche(v)){
167                      return false;
168                  }
169             }
170             //setzen der Werte des Folgeknotens und rekursiver Aufruf ↩
                    der dfs
171             tempZeit = DFS_Zwischenspeicher[k].pfadLaufzeit + k->↩
                    getLaufzeitEinzelgatter();
172             DFS_Zwischenspeicher[v].pfadLaufzeit = tempZeit;
173             DFS_Zwischenspeicher[v].vaterElement = k;
174             if(!dfs_visit(v, s, pfad)){
175                 //wenn Zyklus gefunden
176                 return false;
177             }
178 
179         }
180     }
181 
182     tempZeit = DFS_Zwischenspeicher[k].pfadLaufzeit + k->↩
            getLaufzeitEinzelgatter();
183 
184     //wenn Knoten mit Ausgang verbunden ist, ist neuer Ausgangspfad ↩
            gefunden
185     if(k->getIsAusgangsElement() && laufzeitAusgangspfad < tempZeit){
186         laufzeitAusgangspfad = tempZeit;
187         ausgangspfad = pfad;
188     }
189 
190     return true;
191 
192 }
193 
194 
195 /*
196  * Berechnet die Laufzeit der Einzelgatter und speichert sie im
197  * SchaltwerkElement ab
198  */
199 void LaufzeitAnalysator::berechneLaufzeitEinzelgatter(){
200     ListenElement* li = startElement;
```

```
201     SchaltwerkElement* sE;
202     double laufzeitEinzelgatter;
203     double spg, temp, prozess;
204     double lastKapazitaet;
205     GatterTyp typ;
206 
207     //Faktoren Spannung, Temperatur, Prozess holen
208     faktoren->getFaktoren(spg, temp, prozess);
209 
210     //alle Elemente durchgehen
211     while(li->getNextElement() != NULL){
212         sE = li->getSchaltwerkElement();
213         typ = *(sE->getTyp());
214 
215         //Lastkapazitaeten der Zielelmente aufaddieren
216         lastKapazitaet=0;
217         for(int i=0; i < sE->getAnzahlNachfolger(); i++){
218             lastKapazitaet += sE->getNachfolger(i)->getTyp()->↩
                    getLastKapazitaet();
219         }
220 
221         //Laufzeit berechnen
222         laufzeitEinzelgatter = typ.getGrundLaufzeit() * 1000;    //↩
                *1000 wegen Einheit, piko in femto
223         laufzeitEinzelgatter += typ.getLastFaktor() * lastKapazitaet;
224         laufzeitEinzelgatter *= spg * temp * prozess;
225         //Laufzeit im SchaltwerkElement speichern
226         li->getSchaltwerkElement()->setLaufzeitEinzelgatter(↩
                laufzeitEinzelgatter);
227         //Pointer aufs nächste Element setzen
228         li = li->getNextElement();
229     }
230 }
231 
232 
233 /*
234  * Gibt zurueck, ob in dem Graphen mindestens 1 FlipFlops vorhanden ↩
        ist.
235  */
236 bool LaufzeitAnalysator::flipflopsVorhanden(){
237     ListenElement* element = startElement;
238     //alle ELemente durchgehen
239     while(element->getNextElement() != NULL){
240         if(element->getSchaltwerkElement()->getTyp()->getIsFlipflop())↩
                {
241             return true;
242         }
243         element = element->getNextElement();
244     }
245 
246     return false;
247 }
```

Anhang B

Material und Methoden

B.1 Softwaretools

Programmierumgebung

Da sowohl auf Microsoft Windows als auch auf Apple Mac OSX entwickelt wurde, fand die Plattformübergreifende IDE NetBeans (Version 7.2.1) Verwendung. Diese umfasst neben gängigen Debug-Features auch Möglichkeiten der Code-Vervollständigung, Syntaxhervorhebungen und eine Mercurial-Integration. Das Kompilieren des Programms erfolgte mit dem Apple-LLVM-Compiler.

Versionsverwaltung

Um gleichzeitig an dem Programm arbeiten zu können, wurde die Versionsverwaltungssoftware *Mercurial* verwendet. Diese bietet eine ausführliche Historie der Codeentwicklung und ermöglicht so z.B. jederzeit das Zurückspringen auf eine frühere Version einer Datei. Außerdem sind so die vollzogenen Änderungen im Code sehr übersichtlich nachvollziehbar.

UML-Diagramme

Die UML-Diagramme wurden mit der Software LibreOffice Draw erstellt.

B.2 Hilfsmittel

Als Hilfsmittel wurde die C++ Referenz unter http://www.cplusplus.com/reference/ sowie die Programmierrichtlinien des ITIV benutzt.