

# Dokumentation zum Projektpraktikum Informationstechnik

Gruppe: 064  
Gruppenmitglieder: Benedikt Braunger, Cornelius Richt,  
Fabian Schackmar, Lukas Mohrbacher  
Tutor: Florian Brauchle  
Abgabetermin: 18.01.2013  
Semester: WS2012/2013

# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>3</b>
<b>2</b>	<b>Aufgabenstellung</b>	<b>4</b>
<b>3</b>	<b>Umsetzung</b>	<b>6</b>
3.1	Zeitplanung . . . . .	6
3.2	Materialien . . . . .	6
3.2.1	Programmierungsumgebung . . . . .	6
3.2.2	Versionsverwaltung . . . . .	6
3.2.3	Dokumentation . . . . .	6
3.3	Besondere Codeabschnitte . . . . .	7
3.3.1	Allgemein . . . . .	7
3.3.2	Klasse Äußere Faktoren (faktoren.h/ *.cpp) . . . . .	7
3.3.3	SignalListeErzeuger . . . . .	8
3.3.4	Bibliothek . . . . .	9
3.3.5	Grapherzeuger . . . . .	10
3.3.6	Analyse und Tiefensuche . . . . .	10
3.3.7	Plattform unabhängigkeit . . . . .	12
3.4	Abschlusstest . . . . .	14
<b>4</b>	<b>Fazit</b>	<b>15</b>
<b>A</b>	<b>Quelltext</b>	<b>16</b>
A.1	main.cpp . . . . .	16
A.2	Menue.h . . . . .	17
A.3	Menue.cpp . . . . .	18
A.4	Faktoren.h . . . . .	25
A.5	Faktoren.cpp . . . . .	27
A.6	Bibliothek.h . . . . .	32
A.7	Bibliothek.cpp . . . . .	33
A.8	GatterTyp.h . . . . .	41
A.9	GatterTyp.cpp . . . . .	42
A.10	Flipflop.h . . . . .	44
A.11	Flipflop.cpp . . . . .	45
A.12	SignalListeErzeuger.h . . . . .	46
A.13	SignalListeErzeuger.cpp . . . . .	47
A.14	signals.h . . . . .	52
A.15	signals.cpp . . . . .	53
A.16	SchaltwerkElement.h . . . . .	55
A.17	SchaltwerkElement.cpp . . . . .	57
A.18	ListenElement.h . . . . .	59
A.19	ListenElement.cpp . . . . .	60
A.20	GraphErzeuger.h . . . . .	61
A.21	GraphErzeuger.cpp . . . . .	63
A.22	LaufzeitAnalysator.h . . . . .	71
A.23	LaufzeitAnalysator.cpp . . . . .	72
A.24	cross-compatibility.h . . . . .	77
A.25	cross-compatibility.cpp . . . . .	78

# Kapitel 1

## Einleitung

Im folgenden wird die Umsetzung eines Programmes für die CoolSoft AG im Rahmen des Informationstechnik-Praktikums für den Studiengang Elektro- und Informationstechnik beschrieben.

Es wird als erstes auf die allgemeine Problemstellung eingegangen. Nachfolgend beschreiben wir dann Besonderheiten unserer Lösung und Probleme die während des bearbeiten aufgetreten sind. Schlussendlich ist dann natürlich noch der komplette Code abgedruckt.

In einem umfangreichen Praktikumpaket wurde der genaue Programmaufbau erläutert, inklusive Klassendiagramm. Für einzelne Programmteile wie zum Beispiel die Analyse und Tiefsuche wurde Pseudocode vorgegeben. Somit ging es im allgemeinen nur um die Realisation in C++.

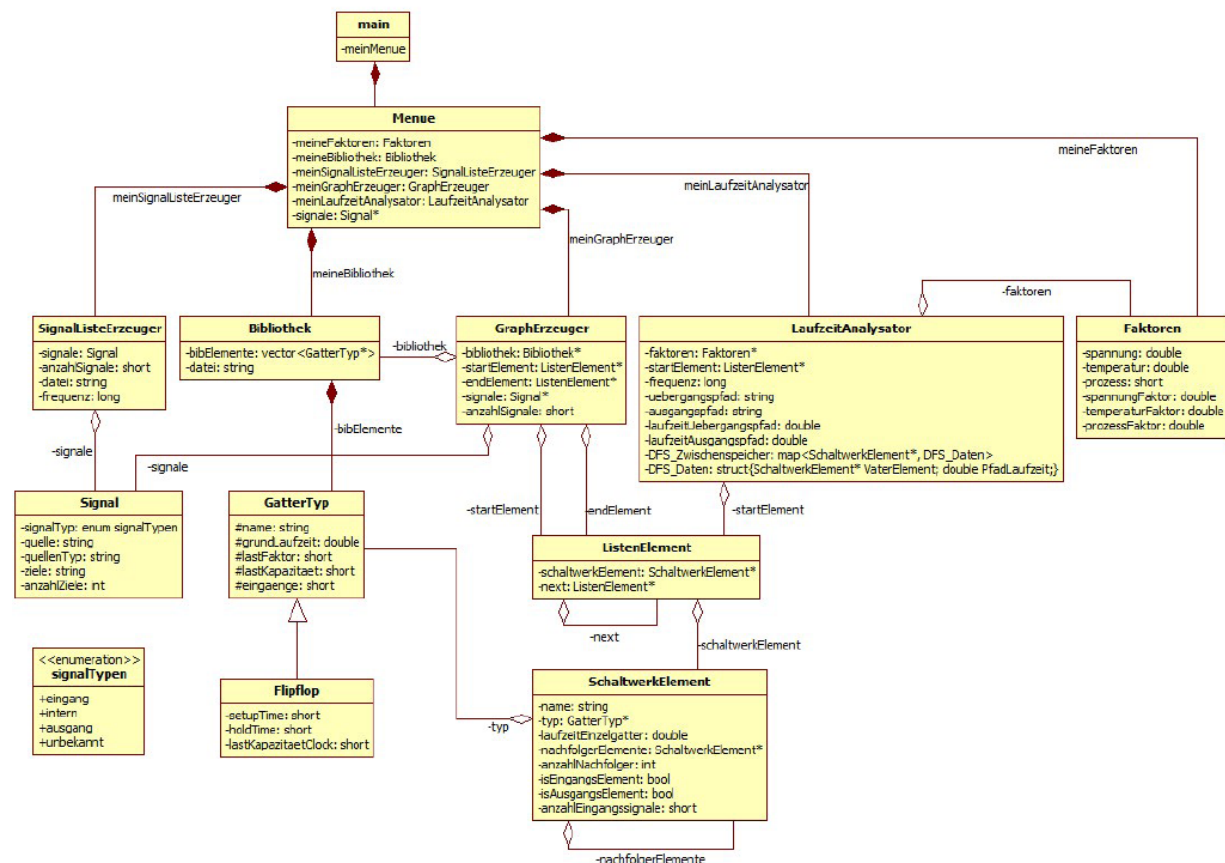


Abbildung 1.1: Klassendiagramm

## Kapitel 2

# Aufgabenstellung

In der gegebenen Aufgabe, soll ein Schaltnetz, welches in Form einer Schaltnetzdatei gegeben ist, analysiert und auf Fehler untersucht werden. Dann soll die maximal mögliche Frequenz berechnet werden. Das Programm soll dabei in der C++ Programmiersprache umgesetzt werden. Um die gewünschten Funktionen ermöglichen zu können, muss zunächst die gegebene Bibliotheksdatei, welche spezifische Informationen über mögliche Schaltwerkelemente enthält, ausgelesen werden und die Informationen in eine passende Datenstruktur überführt werden. Zusätzlich müssen äußere Einflüsse auf die Laufzeit (Betriebsspannung, Temperatur und Herstellungsprozess) in Faktoren umgerechnet werden. Anschließend kann die eigentliche Schaltnetzdatei, welche die Schaltnetzstruktur in Form von Signalen mit entsprechenden Quellen und Zielen darstellt, ausgewertet werden. Auch hier müssen die Informationen in einer geeigneten Datenstruktur gespeichert werden. Erst jetzt kann die eigentliche Funktion des Programms ausgeführt werden. Dazu wird mit Hilfe der Signale ein Schaltnetzgraph erstellt, welcher mit Hilfe des Tiefensuchalgorithmus in einen Tiefenbaum überführt wird. Damit kann dann die maximale Frequenz berechnet werden. Darüber hinaus wurde gefordert, das gesamte Programm komfortabel über das Windows Command Window steuern zu können.

Jedes reale Bauteil weist eine gewisse zeitliche Verzögerung zwischen Ein- und Ausgangssignal auf. Für den vorliegenden Fall in Abbildung 2.1 bedeutet dies, dass das Ausgangssignal des 1. Flipflops eine gewisse Zeit braucht, bis es im Und-Gatter verarbeitet wurde und am Eingang des 2. Flipflops anliegt.

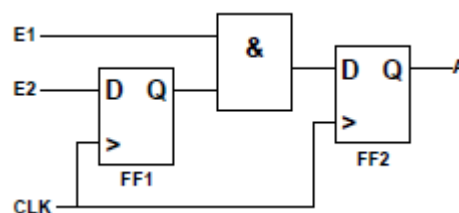


Abbildung 2.1: Einfaches Schaltwerk

Daraus folgt, dass das zweite Flipflop erst dann getriggert werden darf, wenn das Signal dort sicher angekommen ist. Es gibt also eine maximale Taktfrequenz mit der die Schaltung betrieben werden darf. Diese zu finden ist Aufgabe des Programms. Dazu muss das Programm den längsten Signalpfad finden und dessen Laufzeit berechnen. Die Laufzeit hängt noch von diversen äußeren Einflüssen wie Temperatur und Spannung ab, die ebenfalls berücksichtigt werden sollen.

Dieser Rückwärtszähler stellt im Projektpraktikum das Beispielschaltwerk dar, welches untersucht werden soll.

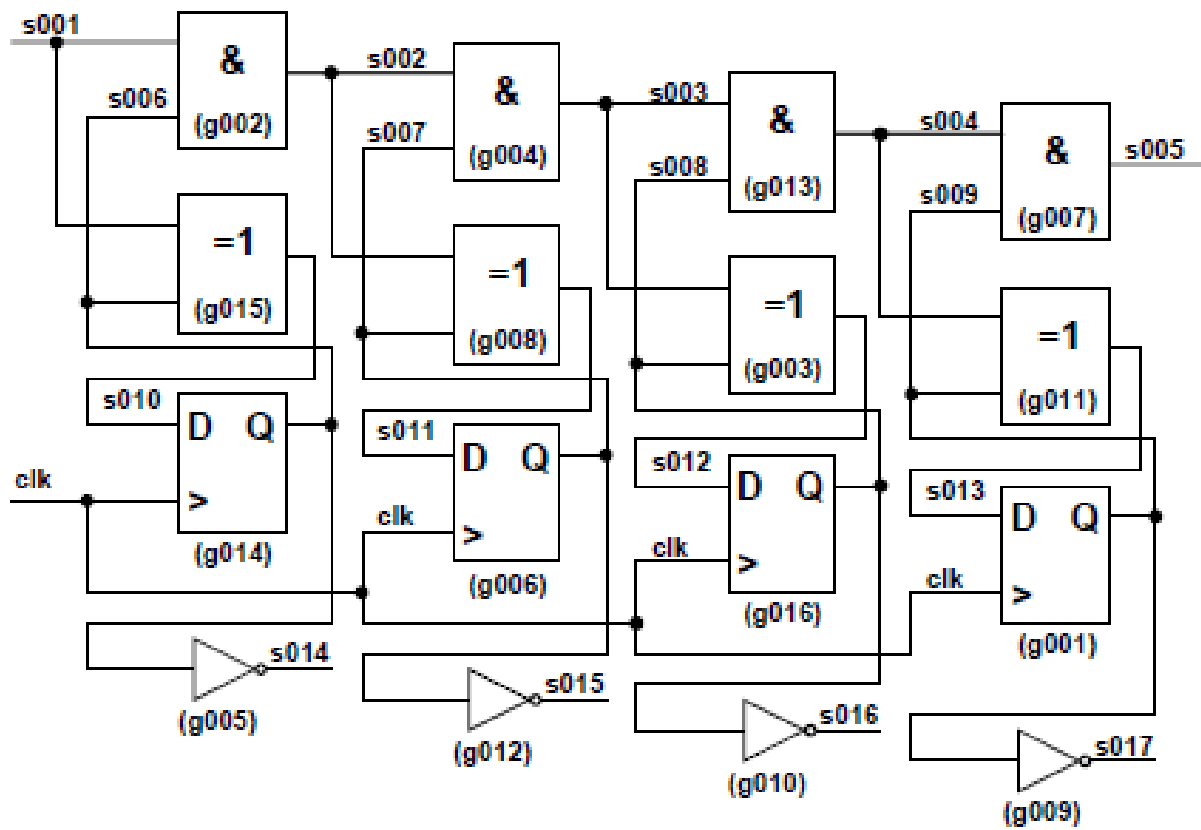


Abbildung 2.2: Rückwärtszähler

## Kapitel 3

# Umsetzung

### 3.1 Zeitplanung

Im ersten Tutorium verschafften wir uns einen groben Überblick über die Aufgabenstellung und besprachen die Vorgehensweise. Die Umsetzung folgte in den nächsten Tutorien bis zum Abschlusstest und der Präsentation vor dem Tutor. Wie an der Git-Grafik zu sehen ist wurde die Möglichkeit donnerstags an der Uni während des Tutoriums zu arbeiten intensiv und fast ausschließlich genutzt.

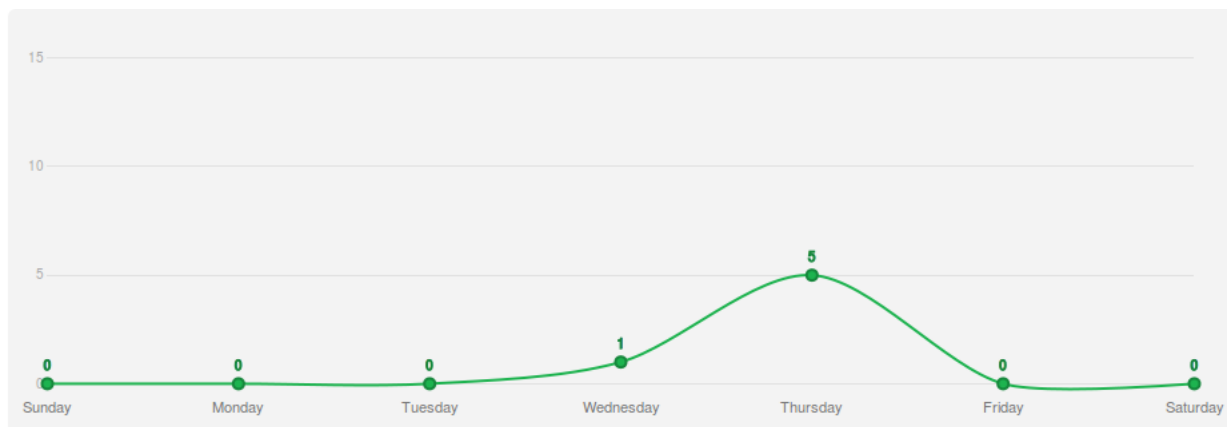


Abbildung 3.1: Git: Durchschnittliche Commits nach Wochentag

### 3.2 Materialien

#### 3.2.1 Programmierumgebung

Als Programmierumgebung wurde hauptsächlich **Code::Blocks** verwendet, da ein Großteil der Gruppe Linux nutzt und dort kein Visual Studio läuft. Kompiliert wurde mit dem gcc, da dieser Compiler unter Linux, wie auch Windows mit nahezu dem gleichen Quellcode auskommt. Mehr dazu im Abschnitt Kompatibilität.

#### 3.2.2 Versionsverwaltung

Der Austausch des Fortschrittes untereinander wurde mit der Versionsverwaltung **Git** gemacht. Als Zentrales Repository wurde der kostenlose Service von Github genutzt. Git ermöglicht es mit geringem Aufwand den Quellcode untereinander synchron zu halten und gleichzeitig auch eine History für Änderungen am Quellcode zu haben.

#### 3.2.3 Dokumentation

Zur Dokumentation verwendeten wir **TeX**. Denn dies spielt mit der Versionsverwaltung Git am besten zusammen und es ist keine aufwendige Formatierung des Dokumentes nötig.

## 3.3 Besondere Codeabschnitte

### 3.3.1 Allgemein

Bibliothek, Gattertyp, Flipflop, Signale und Listenelement sind prinzipiell nur Umsetzung der Prototypen in der Aufgabenstellung bzw. des angegebenen Pseudocodes. Code-Abschnitte die etwas spezieller sind oder von der Aufgabenstellung abweichen sind in diesem Abschnitt erläutert.

### 3.3.2 Klasse Äußere Faktoren (faktoren.h/ \*.cpp)

Diese Klasse beinhaltet Attribute und Methoden, um aus den physikalischen Randbedingungen der Schaltung Faktoren zu errechnen, die in der Laufzeitanalyse später gebraucht werden. Alle Attribute werden als private angelegt und man kann nur mittels der für die 3 Attribute Spannung, Temperatur und Prozess geschriebenen öffentlichen get- und set-Methoden darauf zugreifen. Die 3 Faktorattribute werden in den nach ihnen benannten privaten Methoden (berechneXFaktor(..)) mittels der in ihnen als zweidimensionale Arrays gespeicherten Listen, die 3 (Prozess: slow, typical, fast), 7 (Spannung: 1,08 - 1,32V) oder 15 (Temperatur: -25 - 125°C) Werten einen Faktor zuweisen, berechnet.

Diese schließen zuerst höhere bzw. niedrigere Eingaben als die zugelassenen, durch Ausgabe einer Fehlermeldung aus, und rufen dann allesamt die private Methode "berechneFaktor()" auf, der sie den Array, die Array-Zeilenzahl und den Wert der Randbedingung übergeben. Die Methode durchsucht das übergebene Array vom kleinsten Eintrag an nach dem übergebenen Wert, wobei sie für jeden Listeneintrag (im Array), den der Wert überbietet, eine untere Schranke und dessen direkter Nachfolger in der Liste als obere Schranke festlegt.

Existiert der gesuchte Wert in der Liste wird sofort der diesem zugeordnete Faktor zurückgegeben, wenn nicht, wird die private Methode interpolation(..) mit dem Wert, der unteren und oberen Schranke, sowie deren zugeordnete Faktoren als Parametern aufgerufen. Diese berechnet aus den 4 Schrankenwerten (2 Punkte) eine Geradengleichung ( $m \cdot x + n$ ) und kann somit dem Wert den gesuchten Faktor zurückgeben. Des Weiteren ist noch eine öffentliche Displayausgabefunktion (ausgabeFaktoren()) für die errechneten Faktoren implementiert und öffentlich get-Methode, die alle 3 berechneten Faktoren per Referenzzübergabe zurückgibt (getFaktoren()). Der Konstruktor initialisiert sämtliche Attribute mit 0.

Zum Testen wurde eine eigene Main-Datei geschrieben (FaktorenTestMain.cpp), die zuerst eine Klasse vom Typ äußere Faktoren anlegt. Dann werden via Konsoleneingabe die 3 äußeren Randbedingungen Spannung, Temp. und Prozess übergeben, über die set-Methoden gesetzt und zur Kontrolle die Faktoren (alle zu diesem Zeitpunkt 0) und eingegebenen Werte ausgegeben, dann berechnet und im Anschluss werden die Faktoren - nun mit den richtigen Werten - noch einmal ausgegeben. Für jede Bedingung wurden ungültige Werte eingesetzt, die alle abgefangen wurden.

Listing 3.1: FaktorenTestMain.cpp

```

1 // PIT.cpp : Definiert den Einstiegspunkt für die Konsolenanwendung.
2 //
3
4 // #include "stdafx.h"
5 #include "Faktoren.h"
6
7 #include <iostream>
8 using namespace std;
9
10 int _tmain() {
11     //test
12     Faktoren test;
13     double s,t;
14     short p;
15     cout << "spg: " << endl;
16     cin >> s;
17     cout << endl << "tmp: " << endl;
18     cin >> t;
19     cout << endl << "p: " << endl;
20     cin >> p;
21     test.setProzess(p);
22     test.setSpannung(s);
23     test.setTemp(t);

```

```

24
25     test.ausgabeFaktoren();
26
27     cout << test.getProzess() << endl;
28     cout << test.getSpannung() << endl;
29     cout << test.getTemp() <<endl <<endl;
30
31     cout << test.berechneSpannungFaktor( test.spannung) << endl;
32     cout << test.berechneTempFaktor( test.temp) << endl;
33     cout << test.berechneProzessFaktor( test.prozess) << endl;
34
35     test.ausgabeFaktoren();
36     system("pause");
37
38     return 0;
39 }

```

### 3.3.3 SignalListeErzeuger

Diese Klasse liest die Signaldati zeilenweise ein und aufgrund unterschiedlicher Zeilenanfänge werden diese dann interpretiert, im Gegensatz zur Vorgabe wurden die Signale nicht in einem dynamischen Array gespeichert, sondern in einem Vektor, da dieser einiges flexibler ist als ein Array. Die Ergebnisse dieser Funktion wurden vor der Fertigstellung des Programmes mit der Datei Signal\_test\_main.cpp getestet.

Listing 3.2: Signal\_test\_main.cpp

```

1 #include <iostream>
2 #include "signals.h"
3 #include "SignalListeErzeuger.h"
4
5 using namespace std;
6
7 int main()
8 {
9     string longcat;
10    cout << "File name:";
11    cin >> longcat;
12    SignalListeErzeuger* testlist = new SignalListeErzeuger();
13    cout << "Read: " << testlist->getDatei() << endl;
14    cout << "Vector size: " << testlist->getAnzahlSignale() << endl;
15    cout << "Vectorcontent:" << endl;
16    for (int i=0;i<testlist->getAnzahlSignale();i++) {
17        cout << "-----\n";
18        cout << "Nummer: " << i << endl;
19        cout << "Signaltyp: " << testlist->getSignal(i)->getSignalTyp() << endl;
20        cout << "Quelle: " << testlist->getSignal(i)->getQuelle() << endl;
21        cout << "Quellentyp: " << testlist->getSignal(i)->getQuellenTyp() << endl;
22        cout << "Anzahlziele: " << testlist->getSignal(i)->getAnzahlZiele() << endl;
23        for (int i1=0;i1<testlist->getSignal(i)->getAnzahlZiele();i1++) {
24            cout << "-----" << testlist->getSignal(i)->getZiel(i1) << endl;
25        }
26    }
27
28    return 0;

```



29 }

### 3.3.4 Bibliothek

Stellvertretend für alle Menüpunkte ist hier das Bibliotheksmenü abgebildet, der Ablauf ist immer gleich. Erst wird mit der Funktion `menueKopf` der Header ausgegeben, dann die Menüpunkte und dann wird eine Zahl für den jeweiligen Menüpunkt abgefragt. Das ganze kommt noch in eine Endlosschleife, die mit einem anderen Menüpunkt abgebrochen werden kann und fertig ist das Menü.

Listing 3.3: `Menue::bibliothekMenue()`

```

1 void Menue::bibliothekMenue()
2 {
3     /**
4      * Im Untermenü der Bibliothek soll der Pfad zur Bibliotheksdatei ↵
5      * geändert werden können und man
6      * soll sich zur Kontrolle auch die Datei im Menü anzeigen lassen ↵
7      * können. Auch die Klasse Bibliothek
8      * stellt dazu eine Ausgabemethode bereit.
9      */
10    string pf;
11    while(input != "3") {
12        menueKopf();
13        cout << "Untermenue Bibliothek" << endl;
14        cout << "(1) Pfad zur Bibliotheksdatei: " << meineBibliothek.↵
15            getPfad() << endl;
16        cout << "(2) Ausgabe der Bibliotheksdatei" << endl;
17        cout << "(3) Hauptmenue" << endl << endl;
18        cout << "Waehle einen Menuepunkt und bestaetige mit Enter: ";
19
20        getline(cin, input);
21        switch (atoi(input.c_str())) {
22        case 1:
23            cout << "Pfad eingeben: ";
24            cin >> pf;
25            if(!meineBibliothek.pfadEinlesen(pf)){
26                cout << "Fehler beim einlesen!" << endl;
27                cin.get();
28            } else {
29                meineBibliothek.dateiAuswerten();
30                meinGraphErzeuger.setBibliothek(&meineBibliothek);
31            }
32            break;
33        case 2:
34            meineBibliothek.dateiAusgabe();
35            cin.get();
36            break;
37        }
38    }
39    input.clear();
40 }

```

Listing 3.4: `Menue::menueKopf()`

```

1 void Menue::menueKopf()
2 {
3     /**
4      * Gibt den Kopf der Menüs aus. Dieser bleibt in Hauptmenü und allen ↵
5      * Untermenüs gleich.

```

```

5  */
6  clear_screen();
7  cout << " ***** \n *           IT-↵
    Projektpraktikum WS2012/2013 * \n * ↵
                                * \n * Laufzeitanalyse ↵
    synchroner Schaltwerke * \n ↵
    *****" << endl << endl; ↵
8  //Ausgabe des "Headers"
}

```

### 3.3.5 Grapherzeuger

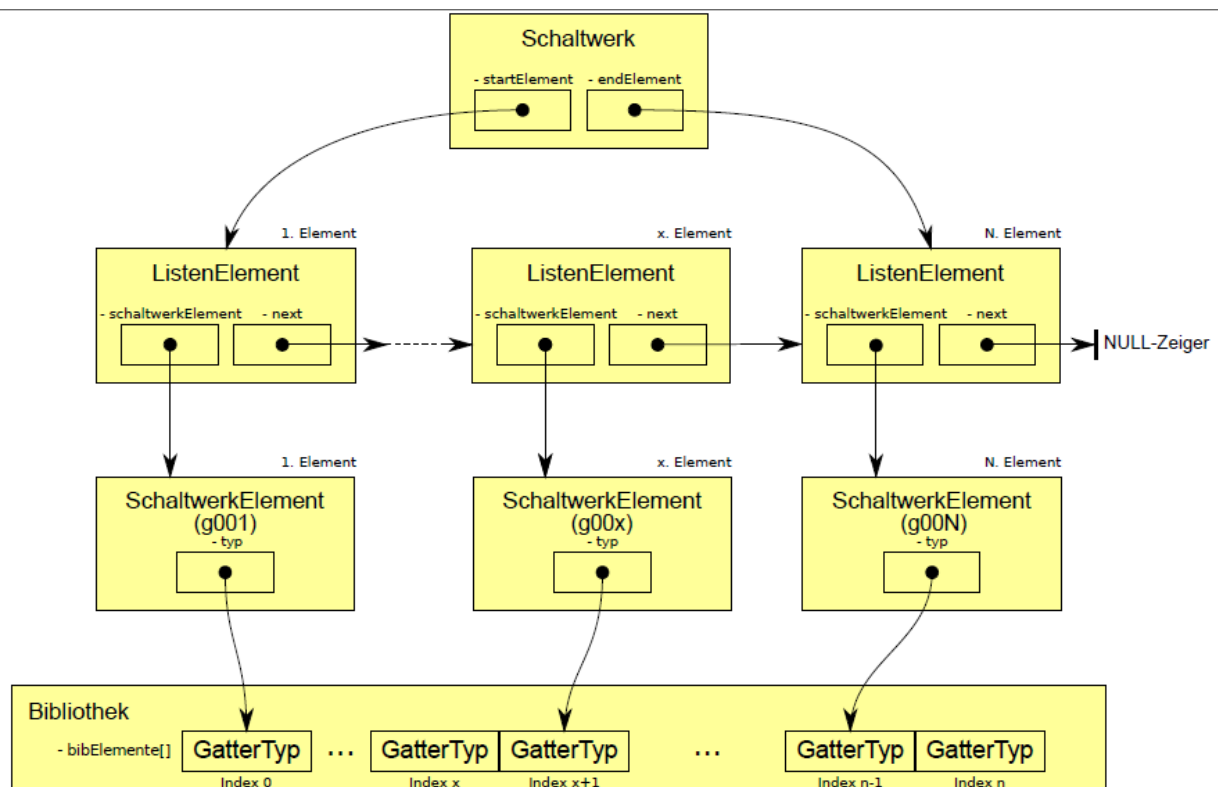


Abbildung 3.2: Schema des Grapherzeuger

### 3.3.6 Analyse und Tiefensuche

#### Beschreibung der Klasse

Die Klasse LaufzeitAnalysator beinhaltet alle notwendigen Methoden und Attribute, die für die Schaltwerk-analyse benötigt werden. Sie benötigt dafür den von der Klasse GraphErzeuger erstellten Graphen. Dieser wird über einen Zeiger auf das erste Listenelement startElement an den LaufzeitAnalysator übergeben. Im Graphen sind bereits alle Informationen der Klasse SignallisteErzeuger und der Klasse Bibliothek verarbeitet. Die zweiten, von außen bereitgestellten Daten, sind die Faktoren und die Frequenz, welche in der Schaltnetzdatei angegeben ist und somit von der Klasse SignallisteErzeuger bereitgestellt wird.

Um die Faktoren auslesen zu können, enthält die Klasse LaufzeitAnalysator einen Zeiger auf das Objekt meineFaktoren. Über diesen Zeiger kann dann in der Methode laufzeitEinzelgatterBerechnen() auf die berechneten Faktoren zugegriffen werden. Die Frequenz wird über eine set-Methode und entsprechender get-Methode der Klasse SignallisteErzeuger gesetzt. Die Klasse steht außerdem in Verbindung mit der Klasse Menuue. Für die Menüintegration werden die entsprechenden Ausgabemethoden bereitgestellt.

#### tiefensuche

Die Funktion tiefensuche() ist die öffentliche Methode der Klasse LaufzeitAnalysator zum Starten des Tiefensuchalgorithmus.

## DFS

Sie hat die Aufgabe den Zwischenspeicher für die Daten der Tiefensuche zurückzusetzen, damit keine Fehler in der Laufzeitberechnung eines Pfades oder in der Abspeicherung des durchlaufenen Pfades passieren. Der Zwischenspeicher wurde, wie in der Spezifikation vorgeschlagen, über eine Map organisiert. Die Map enthält dafür als Schlüsselement einen Zeiger des jeweiligen Knotens vom Typ `SchaltwerkElement*` und als Ziel den struct von Typ `DFS-Daten`. `DFS-Daten` besitzt die Attribute `VaterElement` vom Typ `SchaltwerkElement*`, welches das Vaterelement des Knotens abspeichern soll, welcher als Schlüssel eingegeben wurde. Des weiteren soll der struct die aktuelle Pfadlaufzeit, die bis zum aktuellen Knoten benötigt wurde, speichern. Das dafür benötigte Attribut heißt `PfadLaufzeit` und ist vom Typ `double`. Das Zurücksetzen der Daten wurde über eine for-Schleife durch alle Listenelemente gelöst. Beim ersten Durchlauf wird so die günstige Eigenschaft der Map genutzt, da sie für nicht vorhandene Schlüssel ein neues Wertepaar anlegt. Bei späteren Durchläufen wird ganz normal auf die Map und ihre Wertepaare zugegriffen. Anschließend wird die Methode `DFS-Visit()` mit übergebenen Startknoten aufgerufen.

## DFS-Visit

Die Methode `DFS-Visit()` beschreibt den eigentlichen Tiefensuchalgorithmus. Die Funktion wurde in der Spezifikation der Projektarbeit als Pseudocode gegeben und entsprechend in C++ Code umgesetzt. Die Funktion besitzt zwei Übergabeparameter. Zum einen `SchaltwerkElement*` knoten und zum anderen `SchaltwerkElement*` startknoten. Knoten beschreibt dabei immer den aktuellen Knoten auf dem man sich im Graph befindet. Startknoten ist der Knoten von dem man die Tiefensuche gestartet hat.

Als erstes wird nun eine for-Schleife gestartet in der alle Folgeknoten des aktuellen Knotens durchlaufen werden. In der for-Schleife beendet sich eine if-Struktur in der überprüft wird, ob das aktuelle Nachfolgeelement ein Flipflop ist, oder wenn nicht, ob die Pfadlaufzeit des Nachfolgeknoten kleiner der Pfadlaufzeit bis zum aktuellen Knoten plus der Laufzeit des aktuellen Knotens ist.

Wenn das aktuelle Nachfolgeelement ein Flipflop ist, wird in der nächsten if-Struktur überprüft, ob der Wert des privaten Attributs `laufzeituebergangspfad` kleiner als die Pfadlaufzeit bis zum Knoten knoten und der Laufzeit des Knotens knoten ist. Wenn ja hat der Algorithmus einen neuen längsten Pfad im Überführungsschaltnetz gefunden und die neue längere Laufzeit wird in `laufzeituebergangspfad` abgespeichert. Danach wird ein String der besuchten Gatter erstellt. Dies wird mithilfe einer for-Schleife durchgeführt, die sich iterativ vom aktuellen Knoten bis zum Startelement zurück arbeitet. In der Graphstruktur sind keine Rückwärtskanten vermerkt. Hierzu kann jedoch der `DFS-Zwischenspeicher` genutzt werden, welcher immer den Vaterknoten des entsprechenden Knotens abgelegt hat. Der Name des Knotens, bei welchem sich die for-Schleife beendet, wird damit am Anfang des Strings `uebergangspfad` per `.insert()` eingefügt. Ist das Nachfolgeelement kein Flipflop und die zweite oben beschriebene if-Abfrage greift, wird durch eine weitere if-Abfrage überprüft, ob der Knoten während dieses Durchgangs der Tiefensuche schon einmal besucht wurde. Wenn dies der Fall ist, wird das Vaterelement des Nachfolgeelements auf knoten gesetzt. Danach wird untersucht, ob sich im bisher gegangen Pfad ein Zyklus beendet. Die Zyklensuche mit der entsprechenden Fehlerbehandlung wird im nächsten Abschnitt genauer erklärt, somit soll im Moment davon ausgegangen werden, dass kein Zyklus gefunden wurde. Nach der Zyklensuche wird der Zwischenspeicher mit den neuen Daten für die Laufzeit und dem Vaterelements des Nachfolgeknoten gesetzt und danach rekursiv wieder die Funktion `DFS-Visit()` mit dem Nachfolgeknoten als Knoten aufgerufen. Der Startknoten bleibt unverändert.

Hier sieht man deutlich wie sich der Algorithmus zuerst in die Tiefe vorarbeitet, erst danach in die Breite. Anstatt erst alle Nachfolgeelemente, wie in der for-Schleife ganz am Anfang der Methode gefordert zu durchsuchen, wird rekursiv wieder die Funktion `DFS-Visit()` aufgerufen. Dieser Prozess wird solange wiederholt bis kein Nachfolger mehr vorhanden ist, erst dann wird wieder eine Ebene höher weiter in die Tiefe gesucht.

Wenn der Knoten auf dem man sich gerade beendet keine Nachfolgeelemente besitzt, ist es ein Ausgangselement und somit wird die for-Schleife am Anfang nicht abgearbeitet. Unter der for-Schleife beendet sich der Funktionsteil, den somit alle Ausgangselemente durchlaufen. Als erstes wird durch eine if-Abfrage geprüft ob es sich bei dem Knoten knoten wirklich um ein Ausgangselement handelt und ob die Laufzeit des Knotens plus die Pfadlaufzeit bis zum Knoten größer als die bisher längste Pfadlaufzeit im Ausgangsschaltnetz ist. Wenn das zutrifft wird das Attribut `laufzeitausgangspfad` neu gesetzt und der String `ausgangspfad` analog wie bei `uebergangspfad` erstellt. Wenn die neue Laufzeit nicht größer ist, als die bisher maximale Laufzeit im Ausgangsschaltnetz, wird der Pfad einfach verworfen.

## zyklensuche

Die Zyklensuche wird für jedes neue gefundene Element aufgerufen. Diese Funktion geht vom gegebenen Startelemt über alle Vaterelemente bis sie ganz oben angekommen ist. Sollte der Knoten selbst wieder

gefunden worden sein handelt es sich um einen Zyklus.

### 3.3.7 Plattform unabhängigkeit

Da wir gemischt unter Windows und Linux programmiert haben wollten wir das Programm natürlich nicht nur unter beiden Betriebssystemen kompilieren sondern auch nutzen können. Dabei traten zwei Probleme auf.

#### Konsole löschen

Für den clear Befehl benutzen die beiden Betriebssysteme unterschiedliche Funktionen. Diese sind jedoch in zwei Bibliotheken zu finden und somit musste nur die jeweilige Funktion einfach aufgerufen werden. Um herauszufinden unter welchem System man gerade kompiliert, stellt der compiler eine FLAG zur Verfügung, die man dann ganz einfach auslesen kann. Dieses Auslesen passiert in der Funktion clear\_screen().

Listing 3.5: cross-compatibility.cpp

```
1 #include "cross-compatibility.h"
2 using namespace std;
3
4 void clear_screen() {
5     #if defined _WIN32 || defined _WIN64
6         std::system ( "CLS" );
7     #else
8         // Assume POSIX
9         std::system ( "clear" );
10    #endif
11 }
```

#### Zeilenendung

Windows und Linux definieren die Zeilenendung unterschiedlich, und zwar gibt es zwei Funktionen, zum einen \n für newline und \r für carriage returns, was denn Zeiger an den Anfang der Zeile verschiebt. Windows nutzt nun \n \r hintereinander um ein Zeilenende zu formulieren, unter Linux reicht \r. Nun führt diese unterschiedliche Konvention dazu, dass man unter Windows ein Zeichen zu wenig subtrahiert, wenn man die funktion String.substr() nutzen möchte. Daher der define Linuxzusatz (was aber eigentlich ein Windowszusatz ist).

#### Debug Ausgabe

Um die Ausgabe von debug Nachrichten variabel je nach Debug oder Release zu ermöglichen, gibt es die zwei defines, debug\_msg() und debug\_pause(). Hier wird aufgrund des Defines DEBUG entschieden, ob die Nachricht jetzt ausgegeben werden soll oder nicht.

Listing 3.6: cross-compatibility.h

```
1 // Funktionen um Windows und Linux Kompatibilität zu ermöglichen
2
3 #ifndef CLEARSCREEN_H
4 #define CLEARSCREEN_H
5
6 #include <cstdlib>
7 #include <iostream>
8 #include <stdio.h>
9 #include <stdlib.h>
10 #include <string>
11 using namespace std;
12
13 void clear_screen();
14
15 //void debug_msg(char text);
16
17 #endif
18
19 /// Debug Output
```

```
20     #if defined DEBUG
21         #define debug_msg(text); cout << text << endl;
22     #else //Assume Release
23         #define debug_msg(text);
24     #endif
25
26
27     /// Debug Pause um Debug Output lesen zu können
28     #if defined DEBUG
29         #define debug_pause(); cin.ignore(); cin.get();
30     #else //Assume Release
31         #define debug_pause();
32     #endif
33
34
35 #if defined _WIN32 || defined _WIN64
36     #define linuxzusatz 0
37 #else
38     #define linuxzusatz 1
39 #endif
```

### 3.4 Abschlusstest

Nach der Fertigstellung des Programms testeten wir es mit verschiedenen fehlerhaften Dateien, um eventuelle Kurzschlüsse oder Zyklen im Schaltnetz zu finden und die richtige Fehlermeldung ausgeben zu können. Zudem verglichen wir die ausgegebenen Ergebnisse und prüften sie auf Richtigkeit.

Wir testeten folgende Dateien, welche alle zur richtigen Fehlerausgabe führten

- test Kurzschluss.txt
- test UnbenutztesSignal.txt
- test zyklus.txt
- test OffenerEingang.txt
- test Zyklus1.txt
- Test Schaltnetze.pdf
- test Zyklus2.txt

---

## Kapitel 4

# Fazit

---

Dadurch, dass fast alles in diesem Projekt haargenau vorgegeben war, musste man eine relativ viele Codezeilen einfach nur eins zu eins abtippen und stupide get und set Methoden schreiben. Dies kommt natürlich den Studierenden zu Gute und den Korrektoren wird eine bessere Vergleichsmöglichkeit geboten, allerdings kann man seinen Entwicklergeist nicht entfalten.

---

## Anhang A

# Quelltext

---

### A.1 main.cpp

```
1 #include <iostream>
2 #include "Menue.h"
3
4
5
6 using namespace std;
7
8 int main()
9 {
10     Menue meinMenue;
11     meinMenue.start();
12     return 0;
13 }
```



## A.2 Menue.h

```
1 #ifndef MENUE_H
2 #define MENUE_H
3 #include "SignallisteErzeuger.h"
4 #include <iostream>
5 #include <string>
6 #include "Faktoren.h"
7 #include "Bibliothek.h"
8 #include "cross-compatibility.h"
9 #include "GraphErzeuger.h"
10 #include "LaufzeitAnalysator.h"
11
12 using namespace std;
13
14 class Menue
15 {
16     public:
17         Menue();
18
19         virtual ~Menue();
20
21         void start();
22
23     protected:
24
25
26     private:
27
28         Faktoren meineFaktoren;
29         Bibliothek meineBibliothek;
30         SignallisteErzeuger meinSignallisteErzeuger;
31
32         GraphErzeuger meinGraphErzeuger;
33
34
35
36         void faktorenMenue();
37
38         void bibliothekMenue();
39
40         void schaltwerkMenue();
41
42         void analyse();
43
44         void menueKopf();
45
46         string input;
47
48 };
49
50 #endif // MENUE_H
```

## A.3 Menue.cpp

```

1 #include "Menue.h"
2
3
4
5 /**
6  Menue Klasse
7  zuständig für Ein/Ausgabe und Navigation durch das Programm
8
9  Eine Menüführung innerhalb der Konsole lässt sich am einfachsten ←
    realisieren, indem man zunächst den
10 Inhalt der Konsole liest, die Bildschirmausgabe aktualisiert und ←
    dann die Auswahlmöglichkeiten über
11 eine einfache case-Struktur abfragt.
12 Für die Umsetzung unter Visual Studio können die folgenden Befehle ←
    hilfreich sein:
13 system("pause"); pausiert das Programm bis eine beliebige Taste ←
    gedrückt wird.
14 system("cls"); liest den Inhalt der Konsole.
15
16 Erwünschte Ausgabe:
17 *****
18 *      IT-Projektpraktikum WS2012/2013      *
19 *                                           *
20 * Laufzeitanalyse synchroner Schaltwerke *
21 *****
22
23 (1) äussere Faktoren
24 Spannung [Volt]: 1.2
25 Temperatur [Grad Celsius]: 55
26 Prozess (1=slow, 2=typical, 3=fast): 1
27
28 (2) Bibliothek
29 Pfad zur Bibliotheksdatei: c:\bib.txt
30
31 (3) Schaltwerk
32 Pfad zur Schaltwerksdatei: c:\csd.txt
33
34 (4) Analyse starten
35
36 (5) Programm beenden
37
38 Wähle einen Menüpunkt und bestätige mit Enter:
39 */
40
41
42
43 Menue::Menue()
44 {
45     /**
46      ist der Konstruktor der Klasse. Erzeugt die Objekte meineFaktoren, ←
        meineBibliothek,
47     meinSignalListeErzeuger, meinGraphErzeuger und ←
        meinLaufzeitAnalysator.
48     */
49     Faktoren meineFaktoren;
50     Bibliothek meineBibliothek;
51     SignalListeErzeuger meinSignalListeErzeuger;

```

```

52
53
54
55 //   GraphErzeuger meinGraphErzeuger = new GraphErzeuger;
56 //   LaufzeitAnalysator meinLaufzeitAnalysator = new ←
    LaufzeitAnalysator;
57 //   Signal* signale = new Signal;
58 }
59
60 Menue::~Menue()
61 {
62     /**
63     ist der Destruktor der Klasse.
64     */
65 }
66
67 void Menue::start()
68 {
69     /**
70     schreibt das Hauptmenü in die Konsole und startet die ←
        Hauptschleife, in der durch das Hauptmenü
71     navigiert wird.
72     */
73
74     while(input != "5") {
75         menueKopf();
76
77         /// Faktoren Hauptmenüpunkt
78         cout << "(1) aeussere Faktoren \nSpannung [Volt]: " << ←
            meineFaktoren.getSpannung() << endl;
79         cout << "Temperatur [Grad Celsius]: " << meineFaktoren.getTemp←
            () << endl;
80         cout << "Prozess (1=slow, 2=typical, 3=fast): " << ←
            meineFaktoren.getProzess() << endl;
81
82         cout << endl;
83
84         /// Bibliothek Hauptmenüpunkt
85         cout << "(2) Bibliothek" << endl;
86         cout << "Pfad zur Bibliotheksdatei:" << meineBibliothek.←
            getPfad() << endl;
87
88         cout << endl;
89
90         /// Schaltwerk Hauptmenüpunkt
91         cout << "(3) Schaltwerk \nPfad zur Schaltwerksdatei: " << ←
            meinSignalListeErzeuger.getDatei() << endl;
92
93         cout << "\n(4) Analyse starten \n\n(5) Programm beenden\n\n←
            nWaehle einen Menuepunkt und bestaetige mit Enter: ";
94
95
96         getline(cin, input);
97         switch (atoi(input.c_str()))
98         {
99             case 1:
100                 faktorenMenue();
101                 break;
102             case 2:

```

```

103         bibliothekMenue();
104         break;
105     case 3:
106         schaltwerkMenue();
107         break;
108     case 4:
109         analyse();
110         break;
111     }
112 }
113 }
114
115 void Menue::faktorenMenue()
116 {
117     /**
118     Im Untermenü der Äusseren Faktoren sollen die Aussenbedingungen ←
119     geändert und die daraus resultie-
120     renden Faktoren ausgegeben werden können. Dazu wird von der ←
121     Klasse Faktoren eine Ausgabeme-
122     thode bereitgestellt.
123     */
124     while(input != "5") {
125         menueKopf();
126         cout << "Untermenue Aeussere Faktoren" << endl;
127         cout << "(1) Spannung [Volt]: " << meineFaktoren.getSpannung() ←
128             << endl;
129         cout << "(2) Temperatur [Grad Celsius]: " << meineFaktoren. ←
130             getTemp() << endl;
131         cout << "(3) Prozess (1=slow, 2=typical, 3=fast): " << ←
132             meineFaktoren.getProzess() << endl;
133         cout << "(4) Ausgabe errechneter Faktoren" << endl;
134         cout << "(5) Hauptmenue" << endl << endl;
135         cout << "Waehle einen Menuepunkt und bestaetige mit Enter: ";
136
137         getline(cin, input);
138         switch (atoi(input.c_str())) {
139             case 1:
140                 cout << "Neue Spannung eingeben: ";
141                 getline(cin, input);
142                 if ( (atof(input.c_str()) >= 1.08) && (atof(input.c_str() ←
143                     ()) <= 1.32) ) {
144                     meineFaktoren.setSpannung(atof(input.c_str()));
145                 } else {
146                     cout << "Die Spannung muss zwischen 1.08 und 1.32 ←
147                         liegen" <<endl;
148                     cin.get();
149                 }
150             break;
151             case 2:
152                 cout << "Neue Temperatur eingeben: ";
153                 getline(cin, input);
154                 if ( (atof(input.c_str()) >= -25) && (atof(input.c_str()) ←
155                     <= 125) ) {
156                     meineFaktoren.setTemp(atof(input.c_str()));
157                 } else {
158                     cout << "Die Temperatur muss zwischen -25 und 125 ←
159                         liegen!";
160                     cin.get();
161                 }
162             }
163         }
164     }
165 }

```

```

153         break;
154     case 3:
155         cout << "Neue Prozess Geschwindigkeit eingeben: ";
156         getline(cin, input);
157         if((atoi(input.c_str())<=3) & (atoi(input.c_str()) >= 1)) ←
            {
158             meineFaktoren.setProzess(atoi(input.c_str()));
159         } else {
160             cout << "Es gibt nur 1, 2 und 3!" <<endl;
161             cin.get();
162         }
163         break;
164     case 4:
165         meineFaktoren.ausgabeFaktoren();
166         cin.get();
167         break;
168     }
169 }
170 input.clear();
171 }
172
173 void Menue::bibliothekMenue()
174 {
175     /**
176      Im Untermenü der Bibliothek soll der Pfad zur Bibliotheksdatei ←
177      geändert werden können und man
178      soll sich zur Kontrolle auch die Datei im Menü anzeigen lassen ←
179      können. Auch die Klasse Bibliothek
180      stellt dazu eine Ausgabemethode bereit.
181      */
182     string pf;
183     while(input != "3") {
184         menueKopf();
185         cout << "Untermenue Bibliothek" <<endl;
186         cout << "(1) Pfad zur Bibliotheksdatei: " << meineBibliothek.←
            getPfad() <<endl;
187         cout << "(2) Ausgabe der Bibliotheksdatei" << endl;
188         cout << "(3) Hauptmenue" << endl<< endl;
189         cout << "Waehle einen Menuepunkt und bestaetige mit Enter: ";
190
191         getline(cin, input);
192         switch (atoi(input.c_str())) {
193             case 1:
194                 cout <<"Pfad eingeben: ";
195                 cin >> pf;
196                 if(!meineBibliothek.pfadEinlesen(pf)){
197                     cout << "Fehler beim einlesen!" << endl;
198                     cin.get();
199                 } else {
200                     meineBibliothek.dateiAuswerten();
201                     meinGraphErzeuger.setBibliothek(&meineBibliothek);
202                 }
203                 break;
204             case 2:
205                 meineBibliothek.dateiAusgabe();
206                 cin.get();
207                 break;
208         }
209     }

```

```

208     }
209     input.clear();
210 }
211
212 void Menue::schaltwerkMenue()
213 {
214     /**
215     Im Untermenü des Schaltwerks soll der Pfad zur Schaltwerksdatei ↵
216     veränderbar sein. Zur Kon-
217     trolle soll diese ausgegeben werden können. Ausserdem soll eine ↵
218     Liste der Signale und die Gra-
219     phstruktur ausgegeben werden können. Zu diesen Ausgaben werden ↵
220     Methoden durch die Klassen
221     SignallisteErzeuger und LaufzeitAnalysator bereitgestellt.
222     */
223     while(input != "5") {
224         string pf;
225         menueKopf();
226         cout << "Untermenue Schaltwerk" << endl;
227         cout << "(1) Pfad zur Schaltnetzdatei: " << ↵
228             meinSignallisteErzeuger.getDatei() << endl;
229         cout << "(2) Ausgabe der Schaltnetzdatei" << endl;
230         cout << "(3) Ausgabe der Signale" << endl;
231         cout << "(4) Ausgabe der Graphstruktur" << endl;
232         cout << "(5) Hauptmenue\n\nWaehle einen Menuepunkt und ↵
233             bestaetige mit Enter: ";
234
235         getline(cin, input);
236         switch (atoi(input.c_str())) {
237         case 1:
238             if (meineBibliothek.getPfad() != "") {
239                 cout << "Pfad eingeben: ";
240                 cin >>pf;
241                 ifstream f(pf.c_str());
242                 if(!f.good()) {
243                     cout << "Datei nicht gefunden!"<<endl;
244                     cin.ignore();
245                     cin.get();
246                 } else {
247                     meinSignallisteErzeuger.setDatei(pf);
248                     if (meinSignallisteErzeuger.readFile() == 21 ) {
249                         //cout << "Kurzschluss gefunden!"<< endl;
250                         cin.get();
251                     } else {
252                         debug_pause();
253                         meinGraphErzeuger.listeAnlegen(↵
254                             meinSignallisteErzeuger);
255                         meinGraphErzeuger.graphErzeugen(↵
256                             meinSignallisteErzeuger);
257                     }
258                     debug_pause();
259                 }
260             } else {
261                 cout << "\n Die Bibliothek muss als erstes geladen ↵
262                     werden!";
263                 cin.get();
264             }
265         }
266     }
267 }
268

```

```

259         break;
260     case 2:
261         meinSignalListeErzeuger.dateiAusgabe();
262         cin.get();
263         break;
264     case 3:
265         cout << "Vector size: " << meinSignalListeErzeuger.↵
                getAnzahlSignale() << endl;
266         cout << "Vectorcontent:" << endl;
267         for (int i=0;i<meinSignalListeErzeuger.getAnzahlSignale();↵
                i++) {
268             cout << "↵
                -----\n";
269             cout << "Nummer: " << i << endl;
270             cout << "Signaltyp: " << meinSignalListeErzeuger.↵
                getSignal(i)->getSignalTyp() << endl;
271             cout << "Quelle: " << meinSignalListeErzeuger.↵
                getSignal(i)->getQuelle() << endl;
272             cout << "Quellentyp: " << meinSignalListeErzeuger.↵
                getSignal(i)->getQuellentyp() << endl;
273             cout << "Anzahlziele: " << meinSignalListeErzeuger.↵
                getSignal(i)->getAnzahlZiele() << endl;
274             for (int i1=0;i1<meinSignalListeErzeuger.getSignal(i)↵
                ->getAnzahlZiele();i1++) {
275                 cout << "-----" << meinSignalListeErzeuger.↵
                getSignal(i)->getZiel(i1) <<endl;
276             }
277         }
278         cin.get();
279         break;
280     case 4:
281         meinGraphErzeuger.listenAusgabe( );
282         cin.get();
283         break;
284     }
285 }
286 input.clear();
287 }
288
289 void Menue::analyse()
290 {
291     /**
292     ruft die zur Analyse benötigten Methoden auf und gibt das ↵
     Ergebnis auf dem Bildschirm aus.
293     */
294     if ((meineFaktoren.getTemp() != 0) && (meineFaktoren.getSpannung()↵
        != 0) && (meineFaktoren.getProzess() != 0) && (↵
        meineBibliothek.getPfad() != "") && (meinSignalListeErzeuger.↵
        getDatei() != "")) {
295         LaufzeitAnalysator lza( &meinGraphErzeuger, &meineFaktoren);
296         lza.berechne_LaufzeitEinzelgatter();
297
298         if(lza.DFS_startSuche(&meinGraphErzeuger)){
299
300             lza.maxFrequenz(meinSignalListeErzeuger.getFrequenz());
301         }
302     } else {
303         cout << "Es sind noch nicht alle benötigten Parameter ↵
                ausgefüllt!";

```

```

304     }
305     cin.get();
306     input.clear();
307 }
308
309 void Menue::menueKopf()
310 {
311     /**
312     Gibt den Kopf der Menüs aus. Dieser bleibt in Hauptmenü und ↵
313     allen Untermenüs gleich.
314     */
315     clear_screen();
316     cout << " ***** \n * IT-↵
317             Projektpraktikum WS2012/2013 * \n * ↵
318                                     * \n * Laufzeitanalyse ↵
319             synchroner Schaltwerke * \n ↵
320             *****" << endl << endl; ↵
321     //Ausgabe des "Headers"

```



## A.4 Faktoren.h

```

1 //Faktoren.h
2 //
3 //
4
5
6 #ifndef _Faktoren_
7 #define _Faktoren_
8
9 class Faktoren {
10     private:
11
12     double    spannung,
13             temp,
14             spannungFaktor,
15             tempFaktor,
16             prozessFaktor;
17     short    prozess;
18
19     /** Beinhaltet die Werte Spannungstabelle in Form eines 2-
20         dimensionalen Arrays. Überprüft anhand des Arrays, ob der Wert
21         vom Attribut spannung innerhalb
22         der vorgegebenen Grenzen liegt. Wenn dies nicht der Fall ist, wird
23         eine Fehlermeldung ausgegeben und false zurück gegeben.
24         Ansonsten wird die private Methode
25         berechneFaktor() mit dem Wert, dem Array und der Größe des Arrays
26         als Übergabeparameter aufgerufen. Der Rückgabewert der Methode
27         berechneFaktor() wird in dem
28         Attribut spannungFaktor gespeichert. */
29     bool berechneSpannungFaktor (double spannung);
30
31     /** Beinhaltet die Werte Temperaturtabelle in Form eines 2-
32         dimensionalen Arrays. Überprüft anhand des Arrays, ob der Wert
33         vom Attribut temp innerhalb
34         der vorgegebenen Grenzen liegt. Wenn dies nicht der Fall ist, wird
35         eine Fehlermeldung ausgegeben und false zurück gegeben.
36         Ansonsten wird die private Methode
37         berechneFaktor() mit dem Wert, dem Array und der Größe des Arrays
38         als Übergabeparameter aufgerufen. Der Rückgabewert der Methode
39         berechneFaktor() wird in dem
40         Attribut tempFaktor gespeichert. */
41     bool berechneTempFaktor (double temp);
42
43     /** Beinhaltet die Werte Prozesstabelle in Form eines 2-dimensionalen
44         Arrays. Überprüft anhand des Arrays, ob der Wert vom Attribut
45         prozess innerhalb
46         der vorgegebenen Grenzen liegt. Wenn dies nicht der Fall ist, wird
47         eine Fehlermeldung ausgegeben und false zurück gegeben.
48         Ansonsten wird die private Methode
49         berechneFaktor() mit dem Wert, dem Array und der Größe des Arrays
50         als Übergabeparameter aufgerufen. Der Rückgabewert der Methode
51         berechneFaktor() wird in dem
52         Attribut prozessFaktor gespeichert. */
53     bool berechneProzessFaktor (short prozess);
54
55     /** Die Methode durchsucht das übergebene Array nach dem
56         übergebenen Wert. Wenn der Wert im Array vorhanden ist (1.
57         Spalte der Tabelle) wird der zugehörige

```

```

38  Faktor (2. Spalte der Tabelle) direkt zur  ckgegeben, ansonsten wird
    mit den am n  chsten liegenden Punkten eine Interpolation   ber
    die entsprechende Methode
39  gestartet und der interpolierte Wert zur  ckgegeben. */
40  double berechneFaktor (double wert, double arr[][2], int laenge);
41
42  /** Diese Methode interpoliert einen Wert zwischen zwei vorgegebenen
    Punkten im 2D-Raum. Dabei bestimmen x1,
43  y1 und x2, y2 jeweils die Koordinaten der zwei Punkte zwischen denen
    interpoliert werden soll. Der
44    bergabeparameter wert bestimmt den x-Wert des gesuchten Wertes,
    dabei gilt x1 < wert < x2.*/
45  double interpolation ( double wert, double x1, double x2, double y1,
    double y2);
46
47  public:
48
49  /** Konstruktor der Klasse. Er soll beim Anlegen der Klasse alle
    Attribute mit dem Wert 0 initialisieren.*/
50  Faktoren();
51
52  /** Destruktor der Klasse. */
53  ~Faktoren();
54
55  /** Diese Methode dient zum Lesen des privaten Attributes spannung
    */
56  double getSpannung();
57
58  /** Diese Methode dient zum Lesen des privaten Attributes temp (die
    Temperatur) */
59  double getTemp();
60
61  /** Diese Methode dient zum Lesen des privaten Attributes prozess */
62  short getProzess();
63
64  /** dient zum Lesen (  ber Referenz  bergabe) der entsprechenden
    privaten Attribute. */
65  void getFaktoren(double& spgFaktor, double& tmpFaktor, double&
    przFaktor);
66
67  /** Diese Methode dient zum Schreiben des privaten Attributes
    spannung */
68  void setSpannung (double spannung);
69
70  /** Diese Methode dient zum Schreiben des privaten Attributes temp
    */
71  void setTemp( double temp);
72
73  /** Diese Methode dient zum Schreiben des privaten Attributes
    prozess */
74  void setProzess (short prozess);
75
76  /** Gibt alle berechneten Faktoren auf dem Bildschirm aus. */
77  void ausgabeFaktoren();
78
79 };
80
81
82 #endif // _Faktoren_

```

## A.5 Faktoren.cpp

```

1 // Faktoren.cpp
2 //
3 #include "Faktoren.h"
4 #include <iostream>
5
6 using namespace std;
7
8
9 /** Konstruktor der Klasse. Er soll beim Anlegen der Klasse alle ↵
    Attribute mit dem Wert 0 initialisieren.*/
10 Faktoren::Faktoren() {
11     spannung = 0;
12     temp = 0;
13     prozess = 0;
14     spannungFaktor = 0;
15     tempFaktor = 0;
16     prozessFaktor = 0;
17 }
18
19
20 /** Destruktor der Klasse.*/
21 Faktoren::~Faktoren() {}
22
23 /** Gibt alle berechneten Faktoren auf dem Bildschirm aus. */
24 void Faktoren::ausgabeFaktoren() {
25     cout << endl << "Spannungsfaktor: \t" << spannungFaktor <<
26         endl << "Temperaturfaktor: \t" << tempFaktor << endl <<
27         "Prozessfaktor: \t\t" << prozessFaktor << endl;
28 }
29
30
31 /** dient zum Lesen (Über ReferenzÜbergabe) der entsprechenden ↵
    privaten Attribute. */
32 void Faktoren::getFaktoren(double& spgFaktor, double& tmpFaktor, ↵
    double& przFaktor){
33
34     spgFaktor = spannungFaktor;
35     tmpFaktor = tempFaktor;
36     przFaktor = prozessFaktor;
37 }
38
39
40 /** Beinhaltet die Werte Spannungstabelle in Form eines 2-↵
    dimensionalen Arrays. Überprüft anhand des Arrays, ob der Wert ↵
    vom Attribut spannung innerhalb
41 der vorgegebenen Grenzen liegt. Wenn dies nicht der Fall ist, wird ↵
    eine Fehlermeldung ausgegeben und false zurück gegeben. Ansonsten↵
    wird die private Methode
42 berechneFaktor() mit dem Wert, dem Array und der Größe des Arrays ↵
    als Übergebeparameter aufgerufen. Der Rückgabewert der Methode ↵
    berechneFaktor() wird in dem
43 Attribut spannungFaktor gespeichert.*/
44 bool Faktoren::berechneSpannungFaktor (double spannung){
45     if ((spannung >= 1.08) && (spannung <= 1.32)) {
46         double spgTabelle[][2] = { {1.08 , 1.121557},
47                                     {1.12 , 1.075332},
48                                     {1.16 , 1.035161},

```

```

49         {1.20 , 1.000000},
50         {1.24 , 0.968480},
51         {1.28 , 0.940065},
52         {1.32 , 0.9144822}}};
53
54     spannungFaktor = Faktoren::berechneFaktor(spannung, spgTabelle, 7 ←
55         );
56     return true;
57 } else {
58     cout << endl << "Fehler: Spannung ueber- oder unterschreitet die ←
59         Grenzwerte!" << endl ;
60     return false;
61 }
62 }
63
64
65 /** Beinhaltet die Werte Temperaturtabelle in Form eines 2-←
66     dimensionalen Arrays. Überprüft anhand des Arrays, ob der Wert ←
67     vom Attribut temp innerhalb
68     der vorgegebenen Grenzen liegt. Wenn dies nicht der Fall ist, wird ←
69     eine Fehlermeldung ausgegeben und false zurück gegeben. Ansonsten ←
70     wird die private Methode
71     berechneFaktor() mit dem Wert, dem Array und der Größe des Arrays ←
72     als Übergabeparameter aufgerufen. Der Rückgabewert der Methode ←
73     berechneFaktor() wird in dem
74     Attribut tempFaktor gespeichert. */
75 bool Faktoren::berechneTempFaktor (double temp){
76     if ( (temp >= -25) && (temp <= 125)) {
77         double tempTabelle[][2] = { {-25 , 0.897498},
78             {-15 , 0.917532},
79             { 0 , 0.948338},
80             { 15 , 0.979213},
81             { 25 , 1.000000},
82             { 35 , 1.020305},
83             { 45 , 1.040540},
84             { 55 , 1.061831},
85             { 65 , 1.082983},
86             { 75 , 1.103817},
87             { 85 , 1.124124},
88             { 95 , 1.144245},
89             { 105 , 1.164563},
90             { 115 , 1.184370},
91             { 125 , 1.204966}}};
92
93     tempFaktor = Faktoren::berechneFaktor(temp, tempTabelle, 15 );
94
95     return true;
96 } else {
97     cout << endl << "Fehler: Temperatur ueber- oder unterschreitet die ←
98         Grenzwerte!" << endl ;
99     return false;
100 }
101 }
102 }
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198

```

```

99  /** Beinhaltet die Werte Prozesstabelle in Form eines 2-dimensionalen ↵
    Arrays. Überprüft anhand des Arrays, ob der Wert vom Attribut ↵
    prozess innerhalb
100 der vorgegebenen Grenzen liegt. Wenn dies nicht der Fall ist, wird ↵
    eine Fehlermeldung ausgegeben und false zurück gegeben. Ansonsten ↵
    wird die private Methode
101 berechneFaktor() mit dem Wert, dem Array und der Größe des Arrays ↵
    als Übergabeparameter aufgerufen. Der Rückgabewert der Methode ↵
    berechneFaktor() wird in dem
102 Attribut prozessFaktor gespeichert.*/
103 bool Faktoren::berechneProzessFaktor (short prozess){
104     if ((prozess >= 1) && (prozess <= 3)) {
105         double przTabelle[][2] = { { 1 , 1.174235}, /** 1 = slow */
106                                     { 2 , 1.000000}, /** 2 = typical */
107                                     { 3 , 0.876148}}; /** 3 = fast */
108
109         prozessFaktor = Faktoren::berechneFaktor(prozess, przTabelle, 3 );
110
111         return true;
112     } else {
113         cout << endl << "Fehler: Prozess " << prozess << " existiert nicht ↵
            !! [Slow->1, Typical->2, Fast->3]" << endl ;
114         return false;
115     }
116 }
117 }
118
119
120 /** Die Methode durchsucht das übergebene Array nach dem übergebenen ↵
    Wert. Wenn der Wert im Array vorhanden ist (1. Spalte der Tabelle ↵
    ) wird der zugehörige
121 Faktor (2. Spalte der Tabelle) direkt zurückgegeben, ansonsten wird ↵
    mit den am nächsten liegenden Punkten eine Interpolation über ↵
    die entsprechende Methode
122 gestartet und der interpolierte Wert zurückgegeben. */
123 double Faktoren::berechneFaktor (double wert, double arr[][2], int ↵
    laenge){
124
125     double Faktor;
126     double vgl = 0;
127     double untereSchranke[2] = { (arr[0][0]) , (arr[0][1]) };
128     double obereSchranke[2] = { (arr[laenge-1][0]) , (arr[laenge-1][1]) ↵
        };
129
130     for (int i=0; i < laenge; i++){
131         if (wert > arr[i][0]){
132             untereSchranke[0] = arr[i][0];
133             untereSchranke[1] = arr[i][1];
134             obereSchranke[0] = arr[i+1][0];
135             obereSchranke[1] = arr[i+1][1];
136         }
137         else if (wert == arr[i][0]){
138             vgl = arr[i][0];
139             Faktor = arr[i][1];
140         }
141     }
142
143     if (wert != vgl){

```

```

144     Faktor = Faktoren::interpolation ( wert, untereSchranke[0], ↵
        obereSchranke[0], untereSchranke[1], obereSchranke[1]);
145 }
146 /** cout <<untereSchranke[0]<<endl<< obereSchranke[0]<< endl<<↵
        untereSchranke[1]<<endl<< obereSchranke[1] <<endl; //zum testen ↵
        */
147 return Faktor;
148 }
149
150
151 /** Diese Methode interpoliert einen Wert zwischen zwei vorgegebenen ↵
        Punkten im 2D-Raum. Dabei bestimmen x1,
152 y1 und x2, y2 jeweils die Koordinaten der zwei Punkte zwischen denen ↵
        interpoliert werden soll. Der
153 Ãbergabeparameter wert bestimmt den x-Wert des gesuchten Wertes, ↵
        dabei gilt x1 < wert < x2.*/
154 double Faktoren::interpolation ( double wert, double x1, double x2, ↵
        double y1, double y2){
155     //
156     double interpolwert = wert * (y2-y1)/(x2-x1) + (y1 - (y2-y1)/(x2-x1)↵
        *x1);
157     return interpolwert;
158 }
159
160
161 /** Diese Methode dient zum Lesen des privaten Attributes spannung */
162 double Faktoren::getSpannung(){
163     return spannung;
164 }
165
166
167 /** Diese Methode dient zum Lesen des privaten Attributes temp (die
168 eratur) */
169 double Faktoren::getTemp(){
170     return temp;
171 }
172
173
174 /** Diese Methode dient zum Lesen des privaten Attributes prozess */
175 short Faktoren::getProzess(){
176     return prozess;
177 }
178
179
180 /** Diese Methode dient zum Schreiben des privaten Attributes spannung↵
        */
181 void Faktoren::setSpannung (double spannung){
182     this->spannung = spannung;
183     berechneSpannungFaktor(spannung);
184 }
185
186
187 /** Diese Methode dient zum Schreiben des privaten Attributes temp */
188 void Faktoren::setTemp( double temp){
189     this->temp = temp;
190     berechneTempFaktor(temp);
191 }
192
193

```

```
194 /** Diese Methode dient zum Schreiben des privaten Attributes prozess ↵  
    */  
195 void Faktoren::setProzess (short prozess){  
196     this->prozess = prozess;  
197     berechneProzessFaktor(prozess);  
198 }
```

## A.6 Bibliothek.h

```
1 #ifndef BIBLIOTHEK_H
2 #define BIBLIOTHEK_H
3
4 #include <stdio.h>
5 #include <stdlib.h>
6 #include <iostream>
7 #include <string>
8 #include <vector>
9 #include <fstream>
10 #include "cross-compatibility.h"
11
12
13 #include "GatterTyp.h"
14 #include "Flipflop.h"
15
16
17 using namespace std;
18
19 class Bibliothek{
20 public:
21     Bibliothek();
22     virtual ~Bibliothek();
23
24     string getPfad(void);
25     GatterTyp* getBibElement(string typ);
26     void dateiAusgabe(void);
27     void dateiAuswerten(void);
28     bool pfadEinlesen(string pfad);
29
30 protected:
31 private:
32     string datei;
33     vector<GatterTyp*> bibElemente;
34     vector<Flipflop*> bibHilfe;
35
36     void openError(void);
37     void readError(void);
38 };
39
40 #endif // BIBLIOTHEK_H
```



## A.7 Bibliothek.cpp

```

1
2 #include "Bibliothek.h"
3
4
5
6
7
8
9 Bibliothek::Bibliothek()
10 {
11     vector<GatterTyp*> bibElemente;
12 }
13
14 Bibliothek::~Bibliothek()
15 {
16     //dtor
17 }
18
19 //Dient zum Lesen des Pfads und Dateinamen, welche im Attribut datei ←
    gespeichert sind
20 string Bibliothek::getPfad(void)
21 {
22     return datei;
23 }
24
25 /**Dieser Methode wird ein string, des Gattertyps (z.B. inv1a), ←
    übergeben.
26 Sie gibt einen Zeiger auf das entsprechende Element vom Typ GatterTyp ←
    zurück.
27 */
28 GatterTyp* Bibliothek::getBibElement(string typ)
29 {
30
31     for(int i=0;i< bibElemente.size();i++){
32         if(bibElemente[i]->getName()==typ)
33         {
34             if(bibElemente[i]->getIsFlipflop()){
35                 return (dynamic_cast<Flipflop*>(bibElemente[i]));
36             }
37             else {
38                 return bibElemente[i];
39             }
40         }
41     }
42 }
43
44 }
45
46 /**Ausgabe der Datei auf dem Bildschirm, dabei sollen die Zeilen ←
    durchnummeriert werden.
47 Dabei soll, falls die Datei nicht vorhanden ist oder ein Fehler beim ←
    Lesen auftritt,
48 das Programm nicht absterben, sondern eine Fehlermeldung ausgeben. */
49 void Bibliothek::dateiAusgabe(void)
50 {
51     ifstream f(datei.c_str());
52

```

```

53     string buffer;
54
55     int i=0;
56
57     if(f.good())
58     {
59         while (!f.eof())
60         {
61             getline(f,buffer);
62             cout << i<<" : "<<buffer<<endl;
63             i++;
64         }
65     }
66     else
67     {
68         openError();
69     }
70
71 }
72 /**Die Methode dient zum Einlesen und Auswerten der Bibliotheksdatei.
73 Dabei soll jeder in der Datei beschriebene Gattertyp in einem Element ↵
74 vom Typ
75 GatterTyp im Vektor bibElemente gespeichert werden. Die Reihenfolge ↵
76 ist
77 dabei nicht wichtig. Das Flipflop kann dabei am Namen erkannt werden, ↵
78 welcher als bekannt vorausgesetzt wird.
79 Das Flipflop wird in einem Element vom Typ Flipflop im Vektor ↵
80 bibElemente gespeichert. */
81 void Bibliothek::dateiAuswerten(void)
82 {
83     ifstream f(datei.c_str());
84
85     string buffer;
86     while (!f.eof())
87     {
88         getline(f,buffer);
89         //"\r" entfernen
90         buffer.erase(buffer.size()-1);
91
92         //von [[Bausteine]] bis Leerzeile einlesen
93         if(buffer.find("[[Bausteine]]")==0)
94         {
95
96             while (!f.eof())
97             {
98                 getline(f,buffer);
99
100                 if(buffer=="\r")
101                 {
102                     debug_msg("Blockende gefunden");
103                     break;
104                 }
105
106                 //"\r" entfernen
107                 buffer.erase(buffer.size()-1);

```

```

108         /*if(buffer == "dff")
109         {
110             Flipflop* dummy = (new Flipflop());
111
112             dummy->setName(buffer);
113
114             bibElemente.push_back(*dummy);
115
116             debug_msg( "ff angelegt: "<<buffer);
117
118         }
119         else
120         {
121             GatterTyp* dummy= new GatterTyp();
122             dummy->setName(buffer);
123
124             bibElemente.push_back(*dummy);
125             debug_msg( "gt angelegt: "<<buffer);
126
127         }*/
128
129
130
131     }
132 }
133
134
135 else if(buffer.find("[")==0)
136 {
137     //Klammern [ ] entfernen
138     string name = buffer.substr(1,buffer.size()-2);
139
140
141     //FF anlegen
142     if(name=="dff")
143     {
144         Flipflop *ff = new Flipflop();
145         ff->setName(name);
146
147         debug_msg(name <<"als FF anlegen");
148
149
150         while (!f.eof())
151         {
152             getline(f,buffer);
153
154             //Abbruch falls Absatz zu Ende
155             if(buffer=="\r")
156             {
157
158                 //FF zu bibElemente hinzufügen
159                 bibElemente.push_back((ff));
160
161
162
163
164                 debug_msg("Ende von: "<<name<<" gefunden")←
165                 ;
166                 break;

```

```

166     }
167     //"\r" entfernen
168     buffer.erase(buffer.size()-1);
169
170
171
172     ///*allgemeine Attribute
173     if(buffer.find("ei:")==0)
174     {
175         ff->setEingaenge(atoi(buffer.substr(3).↵
176             c_str()));
177         debug_msg( "ei init " <<ff->getEingaenge())↵
178             ;
179     }
180
181     else if(buffer.find("cl:")==0)
182     {
183         ff->setLastKapazitaet(atoi(buffer.substr(↵
184             3).c_str()));
185         debug_msg("cl init " <<ff->↵
186             getLastKapazitaet());
187     }
188
189     else if(buffer.find("kl:")==0)
190     {
191         ff->setLastFaktor(atoi(buffer.substr(3).↵
192             c_str()));
193         debug_msg("kl init " <<ff->getLastFaktor())↵
194             ;
195     }
196
197     else if(buffer.find("tpd0:")==0)
198     {
199         ff->setGrundLaufzeit(atof(buffer.substr(5)↵
200             .c_str()));
201         debug_msg("tpd0 init " <<ff->↵
202             getGrundLaufzeit());
203     }
204
205     ///*Flipflop Attribute
206     else if(buffer.find("tsetup:")==0)
207     {
208         ff->setSetupTime(atoi(buffer.substr(7).↵
209             c_str()));
210         debug_msg("ff testup init: " <<ff->↵
211             getSetupTime());
212     }
213
214     else if(buffer.find("ed:")==0)
215     {
216         ff->setEingaenge(atoi(buffer.substr(3).↵
217             c_str()));
218         debug_msg("ff ed init: " <<ff->getEingaenge↵
219             ());
220     }
221
222     else if(buffer.find("thold:")==0)
223     {

```

```

213         ff->setHoldTime(atoi(buffer.substr(6).↵
214             c_str()));
215         debug_msg("ff thold init: "<<ff->↵
216             getHoldTime());
217     }
218     else if (buffer.find("cd:")==0)
219     {
220         ff->setLastKapazitaet(atoi(buffer.substr(↵
221             3).c_str()));
222         debug_msg("ff cd init: "<<ff->↵
223             getLastKapazitaet());
224     }
225     else if (buffer.find("tpdt:")==0)
226     {
227         ff->setGrundLaufzeit(atof(buffer.substr(5)↵
228             .c_str()));
229         debug_msg("ff tpdt init: "<<ff->↵
230             getGrundLaufzeit());
231     }
232     else if (buffer.find("kl:")==0)
233     {
234         ff->setLastFaktor(atoi(buffer.substr(3).↵
235             c_str()));
236         debug_msg("ff kl init: "<<ff->↵
237             getLastFaktor());
238     }
239     else if (buffer.find("ct:")==0)
240     {
241         ff->setLastKapazitaetClock(atoi(buffer.↵
242             substr(3).c_str()));
243         debug_msg("ff ct init: "<<ff->↵
244             getLastKapazitaetClock());
245     }
246     else
247     {
248         if (buffer.find("#endif")!=0){
249             //Falls Attribut nicht gefunden
250             readError();
251             debug_msg(buffer<<" nicht gefunden");
252         }
253         else { bibElemente.push_back((ff)); break; }
254     }
255 }
256
257 else {
258
259     GatterTyp *gt = new GatterTyp();
260
261     debug_msg(name <<"als GT anlegen");

```

```

262         gt->setName(name);
263
264
265
266         while (!f.eof())
267     {
268         getline(f,buffer);
269
270         //Abbruch falls Absatz zu Ende
271         if (buffer=="\r")
272         {
273
274             //GT zu bibElemente hinzfügen
275             bibElemente.push_back(gt);
276
277
278
279
280
281             debug_msg("Ende von: "<<name<<" gefunden")<
                ;
282             break;
283         }
284         //"\r" entfernen
285         buffer.erase(buffer.size()-1);
286
287
288
289         ///*allgemeine Attribute
290         if (buffer.find("ei:")==0)
291         {
292             gt->setEingaenge(atoi(buffer.substr(3).<
                c_str()));
293             debug_msg("ei init "<<gt->getEingaenge())<
                ;
294         }
295
296         else if (buffer.find("cl:")==0)
297         {
298             gt->setLastKapazitaet(atoi(buffer.substr<
                (3).c_str()));
299             debug_msg("cl init "<<gt-><
                getLastKapazitaet());
300         }
301
302         else if (buffer.find("kl:")==0)
303         {
304             gt->setLastFaktor(atoi(buffer.substr(3).<
                c_str()));
305             debug_msg("kl init "<<gt->getLastFaktor())<
                ;
306         }
307
308         else if (buffer.find("tpd0:")==0)
309         {
310             gt->setGrundlaufzeit(atof(buffer.substr(5)<
                .c_str()));
311             debug_msg("tpd0 init "<<gt-><
                getGrundlaufzeit());

```

```

312         }
313
314
315         else
316         {
317             if(buffer.find("#endif")!=0){
318                 //Falls Attribut nicht gefunden
319                 readError();
320                 debug_msg(buffer<<" nicht gefunden");
321             }
322             else break;
323         }
324
325     }
326
327 }
328
329 }
330
331 }
332
333
334
335 for(int h=0;h<bibElemente.size();h++){
336     debug_msg( bibElemente[h]->getName());
337 }
338
339
340
341
342
343
344 }
345 /**Speichert den Pfad zu Bibliotheksdatei im entsprechenden Attribut,
346 falls diese unter dem angegebenen Pfad vorhanden ist und sie geändert←
347 werden kann.
348 */
349 bool Bibliothek::pfadEinlesen(string pfad)
350 {
351     ifstream f(pfad.c_str());
352
353     if(f.good())
354     {
355         datei = pfad;
356         return true;
357     }
358     else
359     {
360         openError();
361         return false;
362     }
363 }
364 }
365
366 /**Ausgabe einer Fehlermeldung beim Ändern einer Datei. */
367 void Bibliothek::openError(void)
368 {
369     cerr <<"OPEN ERROR"<<endl;

```

```
370 |
371 | }
372 | /**Ausgabe einer Fehlermeldung beim Lesen einer Datei.
373 | */
374 | void Bibliothek::readError(void)
375 | {
376 |     cerr <<"READ ERROR"<<endl;
377 |
378 | }
```



## A.8 GatterTyp.h

```
1 #ifndef GATTERTYP_H
2 #define GATTERTYP_H
3
4 #include <string>
5
6 using namespace std;
7
8 class GatterTyp
9 {
10     public:
11         GatterTyp();
12         virtual ~GatterTyp();
13
14         string getName(void);
15         double getGrundlaufzeit(void);
16         short getLastFaktor(void);
17         short getLastKapazitaet(void);
18         short getEingaenge(void);
19         virtual bool getIsFlipflop(void);           // muss das nicht ←
20         virtual sein? auch wenn dann sonst was wieder nicht geht..
21         void setName(string n);
22         void setGrundlaufzeit(double gl);
23         void setLastFaktor(short lf);
24         void setLastKapazitaet(short lk);
25         void setEingaenge(short ei);
26
27     protected:
28         string name;
29         double grundlaufzeit;
30         short lastFaktor;
31         short lastKapazitaet;
32         short eingaenge;
33
34     private:
35 };
36
37 #endif // GATTERTYP_H
```

## A.9 GatterTyp.cpp

```
1 #include "GatterTyp.h"
2
3
4 GatterTyp::GatterTyp()
5 {
6     GatterTyp::eingaenge=0;
7     GatterTyp::grundLaufzeit=0;
8     GatterTyp::lastFaktor=0;
9     GatterTyp::lastKapazitaet=0;
10    GatterTyp::name="";
11 }
12
13 GatterTyp::~GatterTyp()
14 {
15     //dtor
16 }
17
18 void GatterTyp::setName(string n)
19 {
20     name=n;
21 }
22
23 string GatterTyp::getName(void)
24 {
25     return name;
26 }
27
28 double GatterTyp::getGrundLaufzeit(void)
29 {
30     return grundLaufzeit;
31 }
32
33 short GatterTyp::getLastFaktor(void)
34 {
35     return lastFaktor;
36 }
37
38 short GatterTyp::getLastKapazitaet(void)
39 {
40     return lastKapazitaet;
41 }
42
43 short GatterTyp::getEingaenge(void)
44 {
45     return eingaenge;
46 }
47
48 bool GatterTyp::getIsFlipflop(void)
49 {
50     return false;
51 }
52
53
54 void GatterTyp::setGrundLaufzeit(double gl)
55 {
56     grundLaufzeit =gl;
57 }
```

```
58|
59| void GatterTyp::setLastFaktor(short lf)
60| {
61|     lastFaktor= lf;
62| }
63|
64| void GatterTyp::setLastKapazitaet(short lk)
65| {
66|     lastKapazitaet=lk;
67| }
68|
69| void GatterTyp::setEingaenge(short ei)
70| {
71|     eingaenge=ei;
72| }
```

## A.10 Flipflop.h

```

1 #ifndef FLIPFLOP_H
2 #define FLIPFLOP_H
3 #include "GatterTyp.h"
4
5 class Flipflop : public GatterTyp
6 {
7     public:
8         Flipflop();
9         virtual ~Flipflop();
10
11         virtual bool getIsFlipflop(void);
12         short getSetupTime(void);
13         short getHoldTime(void);
14         short getLastKapazitaetClock(void);
15         void setSetupTime(short st);
16         void setHoldTime(short ht);
17         void setLastKapazitaetClock(short lkc);
18
19     protected:
20     private:
21         short setupTime;
22         short holdTime;
23         short lastKapazitaetClock;
24
25 };
26
27 #endif // FLIPFLOP_H

```

## A.11 Flipflop.cpp

```
1 #include "Flipflop.h"
2 #include <iostream>
3
4 Flipflop::Flipflop()
5 {
6     holdTime=0;
7     setupTime=0;
8     lastKapazitaetClock=0;
9 }
10
11 Flipflop::~Flipflop()
12 {
13     //dtor
14 }
15
16     bool Flipflop::getIsFlipflop(void){
17
18         return true;
19     }
20     short Flipflop::getSetupTime(void){
21         return setupTime;
22     }
23     short Flipflop::getHoldTime(void){
24         return holdTime;
25     }
26     void Flipflop::setSetupTime(short st){
27         setupTime=st;
28     }
29     void Flipflop::setHoldTime(short ht){
30         holdTime = ht;
31     }
32     void Flipflop::setLastKapazitaetClock(short lkc){
33         lastKapazitaetClock = lkc;
34     }
35     short Flipflop::getLastKapazitaetClock(){
36         return lastKapazitaetClock;
37     }
```

## A.12 SignalListeErzeuger.h

```

1  #ifndef SIGNALLISTEERZEUGER_H
2  #define SIGNALLISTEERZEUGER_H
3
4  #include <iostream>
5  #include <string>
6  #include <sstream>
7  #include <fstream>
8  #include <cstdlib>
9  #include <vector>
10 #include "signals.h"
11 #include "cross-compatibility.h"
12
13 using namespace std;
14
15 /** SignallisteErzeuger
16  Liest die komplette Datei ein, sortiert und erzeugt Liste.
17  Kritik von Lukas: sollte laut Name nur Liste erstellen.
18  */
19
20 class SignallisteErzeuger
21 {
22     public:
23         SignallisteErzeuger();           ///CTOR
24         virtual ~SignallisteErzeuger();  ///DTor
25         Signal* getSignal(int i);         ///gibt Instanz an←
26         int readFile();                   ///Liest Datei ein←
27         int readSignalLine(signalTypen typ, int lengthBegin, string line);
28         int readGateLine(string tmpLine);
29         long getFrequenz();
30         string getDatei();
31         void dateiAusgabe(void);
32         short getAnzahlSignale();
33         void setFrequenz(long freq);
34         void setDatei(string file);       ///Liest Datei ←
35         void setAnzahlSignale(short nSignals);
36     protected:
37     private:
38         vector <Signal> signale;          ///Vector mit ←
39         long frequenz;
40         string datei;                     ///Pfad zur ←
41         short anzahlSignale;
42 };
43
44 #endif // SIGNALLISTEERZEUGER_H

```

## A.13 SignalListeErzeuger.cpp

```

1 #include "SignalListeErzeuger.h"
2
3 /**Konstruktor
4  Setzt alle Variablen auf 0
5  */
6 SignalListeErzeuger::SignalListeErzeuger()
7 {
8     //ctor
9     anzahlSignale = 0;
10    frequenz = 0;
11    datei="";
12    /*setDatei(file);
13    readFile();*/ ///Manuell im Men   aufrufen
14 }
15
16 /**Destruktor
17  */
18 SignalListeErzeuger::~SignalListeErzeuger()
19 {
20     //dtor
21 }
22
23 /**dateiAusgabe
24    ffnet die Datei die in der 'datei' Variable der Klasse gespeichert   
25  ist und gibt die aus
26  */
27 void SignalListeErzeuger::dateiAusgabe(void)
28 {
29     ifstream f(datei.c_str());
30
31     string buffer;
32
33     int i=0;
34
35     if(f.good())
36     {
37         while (!f.eof())
38         {
39             getline(f,buffer);
40             cout << i<<" : "<<buffer << endl;
41             i++;
42         }
43     }
44     else
45     {
46         cout << "ERR: Can not read file!";
47     }
48 }
49 /** Gibt Signal aus dem 'signale' Vektor an der im Parameter   
50  spezifizierten Stelle zur  ck
51  */
52 Signal* SignalListeErzeuger::getSignal(int i) {
53     return &signale.at(i) ;
54 }
55 /**Liest die 'datei' aus und beginnt mit der Auswertung

```

```

56 */
57 int SignallisteErzeuger::readFile() {
58     signale.clear();                                     ///Vektor 'signale' ←
59     wird geleert
60     string line;
61     ifstream listfile(getDatei().data());                ///Öffne ←
62     Dateistream
63     Signal* bufferobj = new Signal;
64     signale.push_back( *bufferobj );                     ///Reserviere ←
65     leeres Objekt für die CLOCK
66     if (listfile.is_open()) {
67         //debug_msg( "INFO: file is open" );
68         while (!listfile.eof()) {
69             getline(listfile,line);                       ///liest ←
70             Zeile für Zeile aus
71             if (((line.substr(0,2)) == "//") or (line == "\r") or (line == "")) {
72                 debug_msg( "INFO: drop, comment or empty line" );
73             }else if ((line.substr(0,12)) == "ARCHITECTURE") { ←
74                 ///Wenn Kommentar, leere ←
75                 Zeile oder Schwachsinn drin steht, passiert gar nichts
76                 debug_msg( "INFO: drop, ARCHITECTURE shit" );
77             }else if ((line.substr(0,6)) == "ENTITY") {
78                 while (1) {
79                     getline(listfile,line);
80                     if ((line.substr(0,2)) == "//") {
81                         debug_msg( "INFO: drop, comment or empty line" ←
82                             );
83                     }else if ((line.substr(0,5)) == "INPUT") {
84                         debug_msg( "INFO: Found INPUT line!" );
85                         readSignalLine(eingang,5,line);
86                     }else if ((line.substr(0,6)) == "OUTPUT") {
87                         debug_msg( "INFO: Found OUTPUT line!" );
88                         readSignalLine(ausgang,6,line);
89                     }else if ((line.substr(0,7)) == "SIGNALS") {
90                         debug_msg( "INFO: Found SIGNALS line!" );
91                         readSignalLine(intern,7,line);
92                     }else if ((line.substr(0,5)) == "CLOCK") {
93                         debug_msg( "INFO: Found CLOCK line!" );
94                         string hr_frequency = line.substr(11,(line. ←
95                             length()-11)); ///←
96                         Schneide Frequenz aus
97                         frequenz = atoi(hr_frequency.data()); ←
98                                     ///←
99
100                        Lese Frequenzzahl
101                        if (hr_frequency.substr(hr_frequency.size() ←
102                            -5,1)=="M") { ←
103
104                            ///Multipliziere frequenz
105                            frequenz = frequenz * 1000000;
106                        } else if (hr_frequency.substr(hr_frequency. ←
107                            size()-5,1)=="k") {
108                            frequenz = frequenz * 1000;
109                        }
110                        bufferobj->setSignalTyp(clk);
111                        signale.at(0) = *bufferobj;
112                        debug_msg( "INFO: Set clk to: " << frequenz ) ←
113                            ;
114                    }else if (line == "\r" or (line == "")){

```



```

98         debug_msg( "INFO: Found empty line, leave ←
           ENTITY area!" );
99         break;
100     }else {
101         debug_msg( "ERR: Error reading line" );
102         break;
103     }
104 }
105 }else if ((line.substr(0,5)) == "BEGIN") {
106     while (1) {
107         getline(listfile,line);
108         if ((line.substr(0,2)) == "//") {
109             debug_msg( "comment" );
110         }else if ((line.substr(0,1)) == "g") {
111             debug_msg( "INFO: Found GATE line!" );
112             if (readGateLine(line) == 1 ) { ←
113
114                 //Wenn Kurzschluss bereits vorhanden ←
115                 cout << "ERR: Short curcuit" << endl;
116                 cin.get();
117                 return 21;
118             }
119         }else if ((line.substr(0,6)) == "END") {
120             debug_msg( "INFO: Found END line!" );
121             signalTypen tmpsig;
122             tmpsig = signale.at(0).getSignalTyp();
123             debug_msg( "DEBUG "<< tmpsig );
124             setAnzahlSignale(signale.size()); ←
125
126             //AnzahlSignale auf die Größe des ←
127             //Vektor setzen
128             debug_msg( "DEBUG: AnzahlSignale: " << ←
129                 getAnzahlSignale() );
130             return 0;
131         }else {
132             debug_msg( "ERR: Error reading line" );
133             break;
134         }
135     }
136 } else {//-----else
137     debug_msg( "ERR: Error reading headline" );
138     break;
139 }
140 }
141 }
142 }
143 int SignallisteErzeuger::readSignalLine(signalTypen typ, int ←
144     lengthBegin, string tmpLine) {
145     string tmpSignal;
146     stringstream tmpStream(tmpLine.substr(lengthBegin+1,(tmpLine.←
147         length()-(lengthBegin+2+linuxzusatz)))); //Erstellt ←
148         Stream und schneidet Anfang und Ende ab

```

```

146 while (getline(tmpStream,tmpSignal','')) { ←
                                                    //////Trennt←
        nach Komma
147 debug_msg( "INFO: Aktuelles Signal: " << tmpSignal );
148 unsigned int tmpSignalNo = atoi(tmpSignal.substr(1,3).c_str())←
        ;                                                    //////Lese Nummer von←
        aktuellem Signal
149 debug_msg( "DEBUG: tmpSignalNo: " << tmpSignalNo );
150 Signal* nullObj = new Signal; ←
                                                    ←
        //////Erzeuge leeres Objekt
151 while (signale.size() <= tmpSignalNo) { ←
                                                    //////Solange ←
        der Vektor kleiner ist als aktuelle Signalnummer
152     signale.push_back( *nullObj ); ←
                                                    ←
        //////Vergrößere Vektor
153 }
154 signale.at(tmpSignalNo).setSignalTyp(typ); ←
                                                    //////Schreibe Typ ←
        an Stelle der akt. Signalnummer in Vektor
155 }
156 }
157
158 int SignallisteErzeuger::readGateLine(string tmpLine) {
159     string gateNo, gatetype, tmpSignal;
160     gateNo = tmpLine.substr(0,4); ←
                                                    //////Schneide Gatenummer ←
        heraus
161     gatetype = tmpLine.substr(5,tmpLine.find("(")-5); ←
                                                    //////Schneide Gatetyp abhängig von der Länge←
        heraus
162     tmpLine = tmpLine.substr(tmpLine.find("(")+1,tmpLine.size()-←
        tmpLine.find("(")-3-linuxzusatz);           //////Schneide ←
        Signale heraus
163     string tmpOut = (tmpLine.substr(tmpLine.size()-2-linuxzusatz,3)); ←
                                                    //////Schneide Ausgang ←
        heraus
164     if (signale.at(atoi(tmpOut.c_str())).getQuelle().empty()) { ←
                                                    //////Prüfe auf Kurzschluss
165         signale.at(atoi(tmpOut.c_str())).setQuelle(gateNo); ←
                                                    //////Setze Quelle ←
        fÄr Ausgangssignal
166     }
167     else {
168         return 1;
169     }
170     signale.at(atoi(tmpOut.c_str())).setQuellentyp(gatetype); ←
                                                    //////Setze Quellentyp ←
        fÄr Ausgangssignal
171     tmpLine = tmpLine.erase(tmpLine.size()-5,5); ←
                                                    //////←
        Schneide Ausgang ab
172     stringstream tmpStream(tmpLine); ←
                                                    //////←
        Erstelle String stream
173     while (getline(tmpStream,tmpSignal','')) { ←
                                                    //////Trenne ←
        nach Komma

```

```
174         debug_msg( "tmpSignal: " << tmpSignal );
175         debug_msg( "DEBUG: Vect: " << tmpSignal.substr(1,3) );
176         if (tmpSignal == "clk") {
177             signale.at(0).zielHinzufuegen(gateNo);
178         }
179         else {
180             signale.at(atoi((tmpSignal.substr(1,3)).c_str())) ←
                zielHinzufuegen(gateNo);           ///F ge Ziele ←
                zu aktuellem Signal hinzu
181         }
182     }
183     return 0;
184 }
185
186 long SignallisteErzeuger::getFrequenz(){
187     return frequenz;
188 }
189 string SignallisteErzeuger::getDatei() {
190     return datei;
191 }
192 short SignallisteErzeuger::getAnzahlSignale(){
193     return anzahlSignale;
194 }
195 void SignallisteErzeuger::setFrequenz(long freq){
196     frequenz = freq;
197 }
198 void SignallisteErzeuger::setDatei(string file){
199     datei = file;
200 }
201 void SignallisteErzeuger::setAnzahlSignale(short nSignals){
202     anzahlSignale = nSignals;
203 }
```

## A.14 signals.h

```
1  /** Signal Class Header File
2  created by Benibr**/
3
4  #ifndef SIGNAL_HEADER
5  #define SIGNAL_HEADER
6
7  #include <string>
8  #include <iostream>
9  #include <vector>
10
11 enum signalTypen {eingang, intern, ausgang, unbekannt, clk};
12
13 using namespace std;
14
15 class Signal
16 {
17     public:
18         Signal();
19         ~Signal();
20         signalTypen getSignalTyp();
21         int getAnzahlZiele();
22         string getQuelle();
23         string getQuellenTyp();
24         string getZiel(int pos);
25         signalTypen setSignalTyp(signalTypen sigTyp);
26         void setQuelle(string gatterName);
27         void setQuellentyp(string gatterTyp);
28         void setAnzahlZiele(int nZiele);
29         void zielHinzufuegen(string gatterno);
30     protected:
31     private:
32         string quelle;
33         string quellenTyp;
34         vector <string> ziele;
35         int anzahlZiele;
36         signalTypen signalTyp;
37 };
38
39
40 #endif
```

## A.15 signals.cpp

```

1  /** Signal Class CPP File
2  created by Benibr */
3
4  #include "signals.h"
5
6  /**Signal() Ist der Konstruktor der Klasse. Er soll beim Anlegen der ↵
   Klasse alle Attribute mit dem Wert 0
7  bzw. NULL fÃ¼r Strings und signalTyp als unbekannt initialisieren.**/
8  Signal::Signal () {
9      quelle = "";
10     quellenTyp = "";
11     //ziele = ;
12     anzahlZiele = 0;
13     signalTyp = unbekannt;
14 }
15 /**Signal() Ist der Destruktor der Klasse. Er soll implementiert ↵
   werden, hat allerdings keine Aufgabe.**/
16 Signal::~Signal () {
17     //dtor
18 }
19 /**type getName(void)
20 Diese Methoden dienen zum Lesen der privaten Attribute eines einzelnen ↵
   Objekts vom Typ Signal.
21 Diese Methoden kÃ¶nnen auch inline implementiert werden.**/
22 int Signal::getAnzahlZiele() {
23     return anzahlZiele;
24 };
25 signalTypen Signal::getSignalTyp() {
26     return signalTyp;
27 }
28 string Signal::getQuelle() {
29     return quelle;
30 };
31 string Signal::getQuellenTyp() {
32     return quellenTyp;
33 };
34 string Signal::getZiel(int pos) {
35     return ziele.at(pos);
36 };
37
38 signalTypen Signal::setSignalTyp(signalTypen sigTyp) {
39     signalTyp = sigTyp;
40 };
41
42 void Signal::setQuelle(string gatterName) {
43     quelle = gatterName;
44 };
45
46 void Signal::setQuellentyp(string gatterTyp) {
47     quellenTyp = gatterTyp;
48 };
49
50 void Signal::setAnzahlZiele(int nZiele) {
51     anzahlZiele = nZiele;
52 };
53
54 void Signal::zielHinzufuegen(string gatterno) {

```

```
55     ziele.push_back(gatterno);  
56     setAnzahlZiele(ziele.size());  
57     //cout << "DEBUG: read form vect@" << anzahlZiele-1 << ": " << ↵  
        ziele.at(ziele.size()-1) << endl;  
58 };
```

## A.16 SchaltwerkElement.h

```

1 // SchaltwerkElement.h
2 //
3 //
4
5 #ifndef _SchaltwerkElement_
6 #define _SchaltwerkElement_
7
8 #include "GatterTyp.h"
9 #include <string>
10
11
12 class SchaltwerkElement {
13 private:
14     string      name;
15     GatterTyp*   typ;
16     double      laufzeitEinzelgatter;
17     SchaltwerkElement* nachfolgerElemente[5];
18     int         anzahlNachfolger;
19     bool         isEingangsElement;
20     bool         isAusgangsElement;
21     short       anzahlEingangssignale;
22
23 public:
24
25     /** Konstruktor der Klasse. Er soll beim Anlegen der Klasse alle ↵
26         Attribute mit dem Wert 0 bzw. NULL für Zeiger initialisieren. ↵
27         Ausserdem
28         bekommt der Konstruktor einen Zeiger auf ein Element der ↵
29         Bibliotheksdatenbank und speichert es in das Attribut typ.*/
30     SchaltwerkElement( GatterTyp* gTyp);
31     ~SchaltwerkElement();           /** Ist der Destruktor der ↵
32         Klasse.*/
33
34     /** Die folgenden Methoden dienen zum Lesen der privaten Attribute ↵
35         eines einzelnen Objekts vom Typ
36         SchaltwerkElement.*/
37
38     string getName();               /** Lesen des privaten ↵
39         Attributes name eines einzelnen Objekts vom Typ ↵
40         SchaltwerkElement. */
41
42     GatterTyp* getTyp();             /** Lesen des privaten ↵
43         Attributes typ eines einzelnen Objekts vom Typ SchaltwerkElement ↵
44         . */
45
46     double getLaufzeitEinzelgatter(); /** Lesen des privaten ↵
47         Attributes laufzeitEinzelgatter eines einzelnen Objekts vom ↵
48         Typ SchaltwerkElement. */
49
50     SchaltwerkElement* getNachfolger( int pos); /** Lesen des privaten ↵
51         Attributes nachfolgerElemente eines einzelnen Objekts vom Typ ↵
52         SchaltwerkElement. */
53
54     int getAnzahlNachfolger();      /** Lesen des privaten ↵
55         Attributes anzahlNachfolger eines einzelnen Objekts vom Typ ↵
56         SchaltwerkElement. */
57
58     short getAnzahlEingangssignale(); /** Lesen des privaten ↵
59         Attributes anzahlEingangssignale eines einzelnen Objekts vom Typ ↵
60         SchaltwerkElement. */
61
62     bool getIsEingangsElement();     /** Lesen des privaten ↵
63         Attributes isEingangsElement eines einzelnen Objekts vom Typ ↵

```

```

    SchaltwerkElement. */
40  bool getIsAusgangsElement();           /** Lesen des privaten ↵
    Attributes isAusgangsElement eines einzelnen Objekts vom Typ ↵
    SchaltwerkElement. */
41
42  /** Die folgenden Methoden dienen zum Schreiben der privaten ↵
    Attribute eines einzelnen Objekts vom Typ
43  SchaltwerkElement.*/
44
45  void setName( string n);               /**Schreiben des privaten ↵
    Attributes name eines einzelnen Objekts vom Typ ↵
    SchaltwerkElement.*/
46  void nachfolgerHinzufuegen( SchaltwerkElement* schaltwerkElement, ↵
    int pos); /**Schreiben des privaten Attributes nachfolger ↵
    eines einzelnen Objekts
47
48
49
50
51
52
53
54 };
55 #endif // _SchaltwerkElement_

```



## A.17 SchaltwerkElement.cpp

```

1 // SchaltwerkElement.cpp
2 //
3 //
4 //
5
6 #include "SchaltwerkElement.h"
7
8
9 /** Konstruktor der Klasse. Er soll beim Anlegen der Klasse alle ↵
    Attribute mit dem Wert 0 bzw Null für Zeiger initialisiern. ↵
    Ausserdem
10 bekommt der Konstruktor einen Zeiger auf ein Element der ↵
    Bibliotheksdatenbank und speichert es in das Attribut typ.*/
11 SchaltwerkElement::SchaltwerkElement( GatterTyp* gTyp ){
12     name = "";
13     typ = gTyp;
14     laufzeitEinzelgatter = 0;
15     nachfolgerElemente[0] = NULL; //schä¶ner l¶¶sen?
16     nachfolgerElemente[1] = NULL;
17     nachfolgerElemente[2] = NULL;
18     nachfolgerElemente[3] = NULL;
19     nachfolgerElemente[4] = NULL;
20     anzahlNachfolger = 0;
21     isEingangsElement = false;
22     isAusgangsElement = false;
23     anzahlEingangssignale = 0;
24 }
25
26
27 SchaltwerkElement::~SchaltwerkElement() { }           /** Destruktor ↵
    der Klasse.*/
28
29 /** Die folgenden Methoden dienen zum Lesen der privaten Attribute ↵
    eines einzelnen Objekts vom Typ
30 SchaltwerkElement.*/
31
32 string SchaltwerkElement::getName(){                   /** Lesen des ↵
    privaten Attributes name eines einzelnen Objekts vom Typ ↵
    SchaltwerkElement. */
33     return name;
34 }
35
36 GatterTyp* SchaltwerkElement::getTyp(){                /** Lesen des ↵
    privaten Attributes typ eines einzelnen Objekts vom Typ ↵
    SchaltwerkElement. */
37     return typ;
38 }
39
40 double SchaltwerkElement::getLaufzeitEinzelgatter(){  /** Lesen des ↵
    privaten Attributes laufzeitEinzelgatter eines einzelnen Objekts↵
    vom Typ SchaltwerkElement. */
41     return laufzeitEinzelgatter;
42 }
43
44 SchaltwerkElement* SchaltwerkElement::getNachfolger( int pos){ /** ↵
    Lesen des privaten Attributes nachfolgerElemente eines einzelnen↵
    Objekts vom Typ SchaltwerkElement. */

```

```

45     return nachfolgerElemente[pos];
46 }
47
48 int SchaltwerkElement::getAnzahlNachfolger(){           /** Lesen des ←
    privaten Attributes anzahlNachfolger eines einzelnen Objekts vom ←
    Typ SchaltwerkElement. */
49     return anzahlNachfolger;
50 }
51
52 short SchaltwerkElement::getAnzahlEingangssignale(){ /** Lesen des ←
    privaten Attributes anzahlEingangssignale eines einzelnen ←
    Objekts vom Typ SchaltwerkElement. */
53     return anzahlEingangssignale;
54 }
55
56 bool SchaltwerkElement::getIsEingangsElement(){           /** Lesen des ←
    privaten Attributes isEingangsElement eines einzelnen Objekts ←
    vom Typ SchaltwerkElement. */
57     return isEingangsElement;
58 }
59
60 bool SchaltwerkElement::getIsAusgangsElement(){           /** Lesen des ←
    privaten Attributes isAusgangsElement eines einzelnen Objekts ←
    vom Typ SchaltwerkElement. */
61     return isAusgangsElement;
62 }
63
64 /** Die folgenden Methoden dienen zum Schreiben der privaten ←
    Attribute eines einzelnen Objekts vom Typ
65     SchaltwerkElement.*/
66 void SchaltwerkElement::setName( string n){
67     name = n;
68 }
69
70 void SchaltwerkElement::nachfolgerHinzufuegen( SchaltwerkElement* ←
    schaltwerkElement, int pos ){
71     //
72     nachfolgerElemente[pos] = schaltwerkElement;
73     //anzahlNachfolger++; //
74     //
75 }
76 void SchaltwerkElement::setAnzahlNachfolger( int anzahlN ){
77     anzahlNachfolger = anzahlN;
78 }
79 void SchaltwerkElement::setAnzahlEingangssignale( short anzahlE ){
80     anzahlEingangssignale = anzahlE;
81 }
82 void SchaltwerkElement::setIsEingangsElement( bool isEingangsEl ){
83     isEingangsElement = isEingangsEl;
84 }
85 void SchaltwerkElement::setIsAusgangsElement( bool isAusgangsEl ){
86     isAusgangsElement = isAusgangsEl;
87 }
88 void SchaltwerkElement::setLaufzeitEinzelgatter( double lzt ){
89     laufzeitEinzelgatter = lzt;
90 }

```

## A.18 ListenElement.h

```
1 // ListenElement.h
2 //
3 //
4
5 #ifndef _ListenElement_
6 #define _ListenElement_
7
8 #include "SchaltwerkElement.h"
9
10
11 class ListenElement {
12 private:
13
14     SchaltwerkElement* schaltwerkElement;
15     ListenElement* next;
16
17 public:
18
19     /** Konstruktor der Klasse. Er soll beim Anlegen der Klasse alle ↵
20         Zeiger-Attribute mit NULL initialisieren.*/
21     ListenElement();
22
23     /** Destruktor der Klasse.*/
24     ~ListenElement();
25
26     /** Lesen des privaten Attributes schaltwerkElement eines einzelnen ↵
27         Objekts vom Typ ListenElement.*/
28     SchaltwerkElement* getSchaltwerkElement();
29
30     /** Lesen des privaten Attributes next eines einzelnen Objekts vom ↵
31         Typ ListenElement.*/
32     ListenElement* getNextElement();
33
34     /** Schreiben des privaten Attributes schaltwerkElement eines ↵
35         einzelnen Objekts vom Typ ListenElement.*/
36     void setSchaltwerkElement( SchaltwerkElement* SchaltwerkEl);
37
38     /** Schreiben des privaten Attributes next eines einzelnen ↵
39         Objekts vom Typ ListenElement.*/
40     void setNextElement( ListenElement* nextEl);
41 };
42
43 #endif // _Listenelement_
```

## A.19 ListenElement.cpp

```

1 // ListenElement.cpp
2 //
3 //
4
5 #include "ListenElement.h"
6
7 /** Konstruktor der Klasse. Er soll beim Anlegen der Klasse alle ↵
8     Zeiger-Attribute mit NULL initialisieren.*/
9 ListenElement::ListenElement(){
10     schaltwerkElement = NULL;
11     next = NULL;
12 }
13
14 /** Destruktor der Klasse.*/
15 ListenElement::~ListenElement() { }
16
17 /** Lesen des privaten Attributes schaltwerkElement eines einzelnen ↵
18     Objekts vom Typ ListenElement.*/
19 SchaltwerkElement* ListenElement::getSchaltwerkElement(){
20     return schaltwerkElement;
21 }
22
23 /** Lesen des privaten Attributes next eines einzelnen Objekts vom Typ ↵
24     ListenElement.*/
25 ListenElement* ListenElement::getNextElement(){
26     return next;
27 }
28
29 /** Schreiben des privaten Attributes schaltwerkElement eines ↵
30     einzelnen Objekts vom Typ ListenElement.*/
31 void ListenElement::setSchaltwerkElement( SchaltwerkElement* ↵
32     SchaltwerkEl){
33     schaltwerkElement = SchaltwerkEl;
34 }
35
36 /** Schreiben des privaten Attributes next eines einzelnen ↵
37     Objekts vom Typ ListenElement.*/
38 void ListenElement::setNextElement( ListenElement* nextEl){
39     next = nextEl;
40 }

```

## A.20 GraphErzeuger.h

```

1 // GraphErzeuger.h
2 //
3 //
4
5 #ifndef _GraphErzeuger_
6 #define _GraphErzeuger_
7
8 // #include "stdafx.h"
9 #include <iostream>
10 #include "SchaltwerkElement.h"
11 #include "ListenElement.h"
12 #include "Bibliothek.h"
13 #include "signals.h"
14 #include "SignallisteErzeuger.h"
15
16
17 class GraphErzeuger {
18 private:
19     Bibliothek* bibliothek;
20     ListenElement* startElement;
21     ListenElement* endElement;
22     Signal* signale;
23     short anzahlSignale;
24
25     int gAnzahl;
26
27 public:
28     GraphErzeuger();           /// Konstruktor; initialisiert alle ↵
29                               /// variablen mit NULL bzw 0
30     ~GraphErzeuger();         /// unnuetzer Destruktor
31
32     void listeAnlegen( SignallisteErzeuger signallist);  /** ↵
33                                                         durchlaeuft die Signalliste, weissst jeder Quelle ein ↵
34                                                         Schaltwerk zu, uebernimmt Eigenschaften der
35                                                         Signale ↵
36                                                         und ↵
37                                                         Gattertypen ↵
38                                                         und ↵
39                                                         verknuepft ↵
40                                                         die ↵
41                                                         Schaltwerke ↵
42                                                         mit ↵
43                                                         je ↵
44                                                         einem ↵
45                                                         Listenelement ↵
46                                                         und ↵
47                                                         die ↵
48                                                         ListenElemente ↵
49                                                         untereinander ↵
50                                                         */
51     void graphErzeugen( SignallisteErzeuger signallist); /** ↵
52                                                         durchlaeuft oben angelegte Liste und verknuepft auf Grundlage ↵
53                                                         der Signalliste die Schaltwerke
54                                                         miteinander ↵
55                                                         */

```

```

36 void listenAusgabe ( ); // bisher nur zum testen /// gibt die ↵
    Liste mit den Schaltwerkinfos aus inkl der von graphErzeugen ↵
    gefundenen Adjazenzbeziehungen
37
38 void setBibliothek( Bibliothek* biblio); /// liest eine ↵
    Bauteilbibliothek ein
39 Bibliothek* getBibliothek(); // gibt die gespeicherte ↵
    Bib zurueck /*(ungebraucht)*/
40
41 ListenElement* getStartElement(); //braucht man nicht
42 void setStartElement( ListenElement* start);
43
44 ListenElement* getEndElement();
45 void setEndElement( ListenElement* ende);
46
47 int getGatterAnzahl(void);
48 };
49 #endif // _GraphErzeuger_

```

## A.21 GraphErzeuger.cpp

```
1 // GraphErzeuger.cpp
2 //
3 //
4
5 #include "GraphErzeuger.h"
6
7 // richtige ausgabe schreiben
8
9 GraphErzeuger::GraphErzeuger()
10 {
11     bibliothek = NULL;
12     startElement = NULL;
13     endElement = NULL;
14     signale = NULL;
15     anzahlSignale = 0;
16 }
17
18
19 /** Destruktor der Klasse.*/
20 GraphErzeuger::~GraphErzeuger()
21 {
22
23 }
24
25 Bibliothek* GraphErzeuger::getBibliothek()
26 {
27     return bibliothek;
28 }
29
30 void GraphErzeuger::setBibliothek( Bibliothek* biblio)
31 {
32     bibliothek = biblio;
33 }
34
35 ListenElement* GraphErzeuger::getStartElement(){
36     return startElement;
37 }
38
39 void GraphErzeuger::setStartElement( ListenElement* start){
40     startElement = start;
41 }
42
43 ListenElement* GraphErzeuger::getEndElement(){
44     return endElement;
45 }
46
47 void GraphErzeuger::setEndElement( ListenElement* ende){
48     endElement = ende;
49 }
50
51 int GraphErzeuger::getGatterAnzahl(){
52     return gAnzahl;
53 }
54
55 void GraphErzeuger::listeAnlegen(SignallisteErzeuger signallist)
56 {
57     startElement = NULL;///Initialisierung
```

```

58     endElement = NULL;
59     signale = NULL;
60     anzahlSignale = 0;
61     gAnzahl = 0;
62
63     short eingaenge = 0;
64     short ausgaenge = 0;
65
66     ListenElement* tmpElement = NULL;
67     anzahlSignale = signallist.getAnzahlSignale();
68
69     /// geht Signalliste durch
70     for (int i = 0; i < anzahlSignale; i++)
71     {
72         Signal tmpSignal = *signallist.getSignal( i );
73
74         if ((tmpSignal.getSignalTyp() == eingang) )           /// prueft ←
75             ob Eingang
76         {
77             debug_msg("INFO: eingang gefunden");
78             eingaenge++;
79         }
80         else if (tmpSignal.getSignalTyp() == clk)           /// prueft ←
81             ob Takt
82         {
83             debug_msg("INFO: clock gefunden");
84         }
85         else if ((tmpSignal.getSignalTyp() == intern) or (tmpSignal. ←
86             getSignalTyp() == ausgang)) /// wenn intern oder ←
87             ausgangs-signal hat das signal eine quelle,
88         {
89             /// die man in Schaltwerke ueberfuehren kann
90             if ( tmpSignal.getQuelle() != "" )           /// zum ←
91                 abfangen von unbenutzten Signalen
92             {
93                 GatterTyp* tmpGatter = bibliothek->getBibElement( ←
94                     tmpSignal.getQuellenTyp()); // kann man sich ←
95                     theoretisch auch sparen und alle tmpGatter durch ←
96                     bibliothek->getBibElement(tmpSignal.getQuellenTyp( ←
97                         ()) ersetzen
98
99                 ListenElement* newListElement = new ListenElement();
100                 SchaltwerkElement* newSchaltwerkElement = new ←
101                     SchaltwerkElement( tmpGatter );
102
103                 /// Schaltwerk uebernimmt Daten des Signals
104                 newSchaltwerkElement->setName(tmpSignal.getQuelle());
105                 newSchaltwerkElement->setAnzahlNachfolger(tmpSignal. ←
106                     getAnzahlZiele());
107                 newSchaltwerkElement->setLaufzeitEinzelgatter( ←
108                     tmpGatter->getGrundLaufzeit() );
109                 newSchaltwerkElement->setAnzahlEingangssignale( ←
110                     tmpGatter->getEingaenge());
111
112                 /// pruefen ob Ausgang
113                 if ( tmpSignal.getSignalTyp() == ausgang )
114                 {
115                     newSchaltwerkElement->setIsAusgangselement(true);
116                     debug_msg("INFO: ausgang gefunden");
117                 }
118             }
119         }
120     }

```



```

104         ausgaenge++;
105     }
106
107     /// verknuepfen von Schaltwerkselement mit ↵
108     Listenelement
109     newListenelement->setSchaltwerkElement( ↵
110         newSchaltwerkElement );
111     gAnzahl++;
112     debug_msg("INFO: " << gAnzahl << ". Listenelement angelegt ↵
113         vom Typ " << tmpSignal.getQuellentyp() << " !");
114
115     /// baut die Liste auf
116     if ( startElement == NULL ) /// ist nur NULL, wenn ↵
117         noch kein Element der Liste existiert
118     {
119         endElement = newListenelement;
120         startElement = newListenelement;
121     }
122     else
123     {
124         tmpElement->setNextElement( newListenelement );
125         endElement = newListenelement;
126     }
127     tmpElement = newListenelement;
128
129     }
130     else /// von leerer Quelle Abfrage, um ungenutzte Signale ↵
131         zu erkennen
132     {
133         cout << "Fehler! Unbenutztes Signal gefunden" << endl;
134         cin.ignore();
135         cin.get();
136     }
137
138     }
139
140     else /// von Signaltypabfrage
141     {
142         cout << "Fehler! Unbekannter Signaltyp" << endl;
143         cin.ignore();
144         cin.get();
145     }
146
147     }
148
149     /// eingang finden
150     for (int z = 0; z < signallist.getAnzahlSignale(); z++) ↵
151         /// geht die Sigalliste durch
152     {
153         if ( signallist.getSignal(z)->getSignalTyp() == eingang) ↵
154             /// vergleicht mit Signaltypen, ob "eingang" ↵
155             der Signaltyp ist
156         {
157             debug_msg( "INFO: Dieses Eingangssignal hat " << signallist. ↵
158                 getSignal(z)->getAnzahlZiele() << " Ziel(e)");
159             for ( int y = 0; y < signallist.getSignal(z)-> ↵
160                 getAnzahlZiele(); y++) /// durchlaeuft ↵
161                 alle ziele dieses signals
162             {

```

```

151         string eingangsGatter = signallist.getSignal(z)->↳
152             getZiel( y );
153         for (ListenElement* ptr = startElement; ptr != NULL; ptr = ptr->getNextElement())    /// und gleicht die
154             ziele mit den schaltwerksnamen in dem
155         {
156             if ( ptr->getSchaltwerkElement()->getName() == eingangsGatter )    ///
157                 jeweiligen listenelement ab
158             {
159                 ptr->getSchaltwerkElement()->↳
160                     setIsEingangsElement(true);
161                 debug_msg( "INFO: "<< ptr->↳
162                     getSchaltwerkElement()->getName() <<" ist
163                     Eingang");
164             }
165         }
166     }
167 }
168
169 /// prueft, ob es unbeschaltete Eingange gibt
170 short tmpZaehler ;
171 for (ListenElement* ptr = startElement; ptr != NULL; ptr = ptr->↳
172     getNextElement())    /// durchlaeuft die Listenelemente
173 {
174     tmpZaehler = 0;
175     debug_msg("INFO:-----");
176     for (int z = 0; z < signallist.getAnzahlSignale(); z++)    ↳
177         ///danach die Signalliste
178     {
179         for (int r = 0; r < signallist.getSignal(z)->↳
180             getAnzahlZiele(); r++)    /// und die Ziele eines
181             jeden Signals
182         {
183             /// prueft, ob das Signalziel mit dem ↳
184             SchaltwerkElementsnamen uebereinstimmt und erhoeht↳
185             den Eingangszaehler bei Erfolg um 1
186             if ( signallist.getSignal(z)->getZiel(r) == ptr->↳
187                 getSchaltwerkElement()->getName())
188             {
189                 tmpZaehler += 1;
190
191                 debug_msg("INFO: "<< tmpZaehler <<". Eingang von "<↳
192                     << ptr->getSchaltwerkElement()->getName() <<" ↳
193                     gefunden");
194             }
195         }
196     }
197 }
198 /// falls dff mit clk, hat die Bib einen Eingang zu wenig, ↳
199 wird hier korrigiert
200 if (ptr->getSchaltwerkElement()->getTyp()->getName()=="dff")    ↳
201     // weil der clock eingang nicht mit eingelesen wird.. ↳
202     sollte man vlt noch aendern
203 {
204     tmpZaehler -= 1;
205 }

```

```

190     }
191     /// check, ob Schaltwerk und Bib fuer das jeweilige Element ←
192         dieselbe Anzahl Eingaenge verzeichnet haben
193     if (ptr->getSchaltwerkElement()->getTyp()->getEingaenge() != ←
194         tmpZaehler)
195     {
196         cout << "Fehler!\nAnzahl Eingaenge laut Bibliothek: \t"<<←
197             ptr->getSchaltwerkElement()->getTyp()->getEingaenge()←
198             <<endl
199         <<"Anzahl Eingaenge laut Schaltwerk: \t"<<tmpZaehler << ←
200             endl;
201         cin.ignore();
202         cin.clear();
203         cin.get();
204     }
205 }
206
207 void GraphErzeuger::graphErzeugen(SignallisteErzeuger signallist)
208 {
209     /// durchlaeuft die Liste der durch ListenElemente verknuepften ←
210         Schaltwerke
211     for (ListenElement* ptr = startElement; ptr != NULL; ptr = ptr->←
212         getNextElement())
213     {
214         ListenElement* tmpListenElement = ptr;
215         SchaltwerkElement* tmpSWE = tmpListenElement->←
216             getSchaltwerkElement();
217
218         /// prueft ob ein Schaltwerk maximal 5 Nachfolger besitzt
219         if ( tmpSWE->getAnzahlNachfolger() <= 5)
220         {
221
222             /// durchlaeuft die Signalliste auf der Suche nach ←
223                 gleichnamigen Quellen der Signale und Schaltwerksnamen
224             for (int i = 0; i < signallist.getAnzahlSignale(); i++)
225             {
226
227                 Signal tmpSignal = *signallist.getSignal( i );
228
229                 if ( tmpSignal.getQuelle() == tmpSWE->getName())
230                 {
231
232                     /// bei Treffer wird wieder die Signalliste ←
233                         durchlaufen auf der Suche nach den Zielen des ←
234                         gleichnamigen Signals
235                     for ( int j = 0; j < tmpSignal.getAnzahlZiele(); j←
236                         ++ )
237                     {
238                         string folgeGatter = tmpSignal.getZiel( j );
239
240                         /// sucht zu den Zielen des Signals das ←
241                             entsprechende Schaltwerk aus den ←
242                             ListenElementen
243                     for (ListenElement* ptr2 = startElement; ptr2 ←
244                         != NULL; ptr2 = ptr2->getNextElement())
245                     {

```

```

234         ListenElement* tmpListenElement2 = ptr2;
235         if ( tmpListenElement2->↳
                getSchaltwerkElement()->getName() == ↳
                folgeGatter )
236         {
237             tmpListenElement->getSchaltwerkElement↳
                (->nachfolgerHinzufuegen( ↳
                tmpListenElement2->↳
                getSchaltwerkElement(), j );
238             debug_msg( "INFO: " << ↳
                tmpListenElement2->↳
                getSchaltwerkElement()->getName() ↳
                << " ist Nachfolger von " << ↳
                tmpListenElement->↳
                getSchaltwerkElement()->getName())↳
                ;
239         }
240     }
241 }
242
243 }
244
245 }
246
247 }
248 else          // von Anzahlnachfolger if-Abfrage
249 {
250     cout << "Fehler: Mehr als 5 Nachfolgegatter bei " << tmpSWE↳
        ->getName() << endl;
251 }
252
253 }
254
255 }
256
257 void GraphErzeuger::listenAusgabe ( )          /// gibt die ↳
        Listenelemente mit Gatternamen und ihre NachfolgeGatter aus
258 {
259
260     for ( ListenElement* ptr = startElement; ptr != NULL; ptr = ptr->↳
        getNextElement())    /// geht die ListenElemente durch
261     {
262
263         cout << "-----\n" << endl
264         << "Gattername:  \t\t" << ptr->getSchaltwerkElement()->getName↳
            () << endl          /// Gattername
265         << "Gattertyp:  \t\t" << ptr->getSchaltwerkElement()->getTyp()↳
            ->getName() << endl;    /// GatterTyp
266
267         /// evtl zusaetzliche Ausgaben wie "Eingang", "Ausgang", "↳
            LaufzeitEinzelGatter", fuer FlipFlops noch andere ↳
            Attribute
268         if ( ptr->getSchaltwerkElement()->getIsEingangsElement() == ↳
            true)          // kann man sich mal noch ueberlegen in die ↳
            Ausgabe mit aufzunehmen
269         {
270             cout << "Schaltungseingangselement" << endl;
271         }

```

```

272     if ( ptr->getSchaltwerkElement()->getIsAusgangselement() == ←
273         true)
274     {
275         cout<<"Schaltungsausgangselement"<< endl;
276     }
277     cout<<"Laufzeit Einzelgatter: \t"<< ptr->getSchaltwerkElement←
278         ()->getLaufzeitEinzelgatter() <<endl;
279
280     cout << "Is Flipflop: \t\t"<< (( Flipflop*) (ptr->←
281         getSchaltwerkElement()->getTyp()) )->getIsFlipflop()<<←
282         endl;
283
284     if (ptr->getSchaltwerkElement()->getTyp()->getName()== "dff")
285     {
286         cout << "Setup-Time: \t\t" << (( Flipflop*) (ptr->←
287             getSchaltwerkElement()->getTyp()) )->getSetupTime() << ←
288             endl
289         << "Hold-Time \t\t" << (( Flipflop*) (ptr->←
290             getSchaltwerkElement()->getTyp()) )->getHoldTime()<<←
291             endl
292         << "Lastkapazitaet: \t"<< (( Flipflop*) (ptr->←
293             getSchaltwerkElement()->getTyp()) )->←
294             getLastKapazitaetClock()<<endl;
295     }
296
297     cout<<"Anzahl Eingangssignale: "<< ptr->getSchaltwerkElement()←
298         ->getAnzahlEingangssignale() <<endl;           // Ende ←
299         Fakultative Ausgabe
300
301     /// Ausgabe der Anzahl der Folgegatter und dann der Gatter mit←
302     ihrem Namen
303     if (ptr->getSchaltwerkElement()->getAnzahlNachfolger()==1){
304         cout<<"--->Das Gatter hat "<< ptr->getSchaltwerkElement()←
305             ->getAnzahlNachfolger()<<" Ziel" <<endl;    /// Einzahl
306     }else if(ptr->getSchaltwerkElement()->getAnzahlNachfolger()←
307         ==0){
308         cout<<"--->Das Gatter hat keine Ziele" <<endl; ←
309                                     //←
310                                     / Keine
311     }else{
312         cout<<"--->Das Gatter hat "<< ptr->getSchaltwerkElement()←
313             ->getAnzahlNachfolger()<<" Ziele" <<endl;    /// Mehrzahl
314     }
315
316     if (ptr->getSchaltwerkElement()->getAnzahlNachfolger()!=0) ///←
317         falls Nachfolger existieren
318     {
319
320         string ausgabe = " ";
321         for ( int s = 0; s < ptr->getSchaltwerkElement()->←
322             getAnzahlNachfolger() ; s++)    /// werden alle ←
323             Nachfolgernamen in einen Ausgabe string geschrieben
324         {
325
326             ausgabe = ausgabe + ptr->getSchaltwerkElement()->←
327                 getNachfolger(s)->getName() + " ";
328         }
329         cout << "Angeschlossene Gatter:\t"<<ausgabe <<endl;
330     }

```

```
309         else /*if (ptr->getSchaltwerkElement()->getAnzahlNachfolger()↵  
310             ==0)*/ /// falls keine Nachfolger existieren  
311         {  
312             cout << ptr->getSchaltwerkElement()->getName() << " hat ↵  
313                 keine Folgegatter"<<endl;  
314         }  
315 }
```

## A.22 LaufzeitAnalysator.h

```

1  #ifndef _LAUFZEITANALYSATOR_H
2  #define _LAUFZEITANALYSATOR_H
3
4  #include <map>
5  #include "ListenElement.h"
6  #include "Faktoren.h"
7  #include "GraphErzeuger.h"
8  #include <vector>
9
10 struct DFS_Daten
11 {
12     SchaltwerkElement* VaterElement;
13     double PfadLaufzeit;
14 };
15
16 class LaufzeitAnalysator
17 {
18 private:
19
20     Faktoren* faktoren;
21     GraphErzeuger* gE;
22
23     long frequenz;
24     string uebergangspfad;
25     string ausgangspfad;
26     double laufzeitUebergangspfad;
27     double laufzeitAusgangspfad;
28     bool zyklusFound;
29     bool zyklensuche(SchaltwerkElement* se);
30     void DFS(ListenElement* s);
31
32     map < SchaltwerkElement* , DFS_Daten > DFS_Zwischenspeicher;
33
34     void DFS_Visit(SchaltwerkElement* k, SchaltwerkElement* s);
35
36
37 public:
38     LaufzeitAnalysator(GraphErzeuger* gE, Faktoren* f);
39     virtual ~LaufzeitAnalysator();
40
41     void berechne_LaufzeitEinzelgatter();
42
43     bool DFS_startSuche(GraphErzeuger* ge);
44     double maxFrequenz(long freq);
45
46 protected:
47
48 };
49
50 #endif // LAUFZEITANALYSATOR_H

```

## A.23 LaufzeitAnalysator.cpp

```

1 #include "LaufzeitAnalysator.h"
2
3
4
5 LaufzeitAnalysator::LaufzeitAnalysator(GraphErzeuger* g, Faktoren* f)
6 {
7
8     faktoren = f;
9     gE = g;
10
11
12
13     laufzeitUebergangspfad=0;
14     laufzeitAusgangspfad=0;
15     DFS_Zwischenspeicher.clear();
16     zyklusFound = false;
17
18 }
19
20 LaufzeitAnalysator::~LaufzeitAnalysator()
21 {
22     //dtor
23 }
24
25 void LaufzeitAnalysator::berechne_LaufzeitEinzelgatter()    ///↵
    berechnet Laufzeit fuer jedes einzelne Gatter
26 {
27     double spgFaktor,
28     tmpFaktor,
29     przFaktor;
30
31     faktoren->getFaktoren(spgFaktor, tmpFaktor, przFaktor);    ///↵
    sich aeussere Faktoren ueber Referenz
32
33     debug_msg( "INFO: SPG-F: "<<spgFaktor<<" TMP-F: "<<tmpFaktor<<"    ///↵
    PRZ-F: "<<przFaktor<<endl<<endl);
34
35     for (ListenElement* ptr = gE->getStartElement(); ptr != NULL; ptr    ///↵
    = ptr->getNextElement())    ///↵ durchlaeuft die ListenElemente    ///↵
    und weist jedem Schaltwerk im Listenelement seine
36     {
37
38         double tpd0 = ptr->getSchaltwerkElement()->getTyp()->    ///↵
    getGrundlaufzeit();    ///↵ Grundlaufzeit tpd0 (in ps)    ///↵
    und seinen
39         double lastFaktor = ptr->getSchaltwerkElement()->getTyp()->    ///↵
    getLastFaktor();    ///↵ Lastfaktor lastFaktor (in fs/fF)    ///↵
    zu
40         double last_C = 0;
41
42         for (int i = 0; i < (ptr->getSchaltwerkElement()->    ///↵
    getAnzahlNachfolger()); i++ )    ///↵ summiert die Eingangs-    ///↵
    C (in fF) der mit dem Ausgang verbundenen Gatter und
43         {
44
45             last_C = last_C + ptr->getSchaltwerkElement()->    ///↵
    getNachfolger(i)->getTyp()->getLastKapazitaet();    ///↵

```



```

    speichert diese im Schaltwerkelement
46     debug_msg("INFO: C-Last: \t"<<last_C<<endl);
47
48     }
49     /// t_pd,actual = (t_pd0 + LastF + LastC) * TempF * SpgF * ←
    PrzF
50     // (ps)          = ( ps  + (fs/fF)  * fF  * 1000)  //wieso ←
    funktioniert mit 1/1000 und nicht mit 1000 ???????
51     ptr->getSchaltwerkElement()->setLaufzeitEinzelgatter(((tpd0 + ←
    lastFaktor * last_C * 0.001) * spgFaktor * tmpFaktor * ←
    przFaktor));  /// berechnet die Gesamtlaufzeit des ←
    Einzelgatters
52
53     debug_msg("INFO: Laufzeit Einzelgatter von "<< ptr->←
    getSchaltwerkElement()->getName() << ":\t"<<ptr->←
    getSchaltwerkElement()->getLaufzeitEinzelgatter()<<endl);
54     }
55
56 }
57 bool LaufzeitAnalysator::DFS_startSuche(GraphErzeuger *gE)
58 {
59     zyklusFound = false;
60     vector < ListenElement *> start;
61
62     for(ListenElement *i = gE->getStartElement(); i!=NULL ; i=i->←
    getNextElement())
63     {
64
65         if(i->getSchaltwerkElement()->getIsEingangselement() or i->←
    getSchaltwerkElement()->getTyp()->getIsFlipflop())
66         {
67
68
69             start.push_back(i);
70         }
71     }
72
73
74
75
76     for(int h=0; h<start.size(); h++)
77     {
78
79         DFS(start[h]);
80     }
81
82
83     return !zyklusFound;
84
85
86
87
88 }
89
90 void LaufzeitAnalysator::DFS(ListenElement* s)
91 {
92
93

```

```

94     for (ListenElement* ptr = s; ptr != NULL; ptr = ptr->
getNextElement())
95     {
96         DFS_Zwischenspeicher[ptr->getSchaltwerkElement()].PfadLaufzeit←
=0.0;
97         DFS_Zwischenspeicher[ptr->getSchaltwerkElement()].VaterElement←
= NULL;
98
99
100    }
101
102    DFS_Visit(s->getSchaltwerkElement(),s->getSchaltwerkElement());
103
104
105 }
106
107 bool LaufzeitAnalysator::zyklensuche(SchaltwerkElement* se)
108 {
109     for (SchaltwerkElement* i=se ; i!=NULL ; i=DFS_Zwischenspeicher[i].←
VaterElement)
110     {
111         if( DFS_Zwischenspeicher[i].VaterElement == se )
112         {
113             return true;
114         }
115     }
116     return false;
117 }
118
119
120
121
122 void LaufzeitAnalysator::DFS_Visit(SchaltwerkElement* k,←
SchaltwerkElement* s)
123 {
124     for (int i=0; i<k->getAnzahlNachfolger(); i++)
125     {
126         if(zyklusFound) {
127             break;
128             debug_msg( "Zyklus gefunde, breche ab!" << endl);
129         }
130
131
132         SchaltwerkElement* v =k->getNachfolger(i);
133         debug_msg("nachfolger:"<<v->getName()<<endl);
134
135         if (v->getTyp()->getIsFlipflop())
136         {
137             debug_msg("ff gefunden: "<<v->getName()<<endl<<endl<<endl)←
;
138
139             if (laufzeitUebergangspfad<DFS_Zwischenspeicher[k].←
PfadLaufzeit + k->getLaufzeitEinzelgatter())
140             {
141                 laufzeitUebergangspfad=DFS_Zwischenspeicher[k].←
PfadLaufzeit + k->getLaufzeitEinzelgatter();
142
143                 uebergangspfad = k->getName() + " ->" + k->←
getNachfolger(i)->getName();

```

```

144         //cout << "Übergangspfad: "<<uebergangspfad<<":"<<↵
145         laufzeitUebergangspfad<<endl;
146         //String erstellen
147         for( SchaltwerkElement* v = k ; v != s ; v = ↵
148             DFS_Zwischenspeicher[v].VaterElement )
149         {
150             uebergangspfad.insert( 0 , ( DFS_Zwischenspeicher[↵
151                 v].VaterElement->getName() + "->" ));
152         }
153     }
154     else if (DFS_Zwischenspeicher[v].PfadLaufzeit < (↵
155         DFS_Zwischenspeicher[k].PfadLaufzeit +k->↵
156         getLaufzeitEinzelgatter()))
157     {
158         if( ( ( DFS_Zwischenspeicher[ v ].PfadLaufzeit != 0 ) or (↵
159             v== s ) ) and ( DFS_Zwischenspeicher[ v ].↵
160             VaterElement != k) )
161         {
162             DFS_Zwischenspeicher[v].VaterElement =k;
163
164             if(zyklensuche(v))
165             {
166                 zyklusFound = true;
167                 cout << "Fehler Zyklensuche!"<<endl;
168             }
169         }
170         DFS_Zwischenspeicher[v].PfadLaufzeit = ↵
171             DFS_Zwischenspeicher[k].PfadLaufzeit + k->↵
172             getLaufzeitEinzelgatter();
173         DFS_Zwischenspeicher[v].VaterElement = k;
174         DFS_Visit(v,s);
175     }
176 }
177
178
179 if(k->getIsAusgangsElement() and (laufzeitAusgangspfad < (↵
180     DFS_Zwischenspeicher[k].PfadLaufzeit + k->↵
181     getLaufzeitEinzelgatter()))
182 {
183     laufzeitAusgangspfad = DFS_Zwischenspeicher[k].PfadLaufzeit +↵
184     k->getLaufzeitEinzelgatter();
185
186     ausgangspfad = k->getName();
187
188     for(SchaltwerkElement * j = k; j != s; j= DFS_Zwischenspeicher↵
189         [j].VaterElement)
190     {
191         ausgangspfad.insert(0,(DFS_Zwischenspeicher[j].↵
192             VaterElement->getName() + "->"));
193     }

```

```

189
190     }
191
192 }
193
194 double LaufzeitAnalysator::maxFrequenz(long freq)
195 {
196
197     cout << "L ngster Pfad im  berfuehrungsschaltnetz:" << endl;
198     cout << uebergangspfad << endl << endl;
199     cout << "Maximale Laufzeit der Pfade im  berfuehrungsschaltnetz: " <<
        " << laufzeitUebergangspfad << " ps" << endl;
200     cout << endl;
201     cout << "L ngster Pfad im Ausgangsschaltnetz:" << endl;
202     cout << ausgangspfad << endl << endl;
203     cout << "Maximale Laufzeit der Pfade im Ausgangsschaltnetz: " <<
        laufzeitAusgangspfad << " ps" << endl;
204     cout << endl;
205     cout << "-----" << endl;
206     long double maxF = 1/ (dynamic_cast<Flipflop*>(( gE->getBibliothek
        (->getBibElement("dff"))->getSetupTime() * 0.000000000001 +
        laufzeitUebergangspfad * 0.000000000001 ));
207
208     if(maxF>1000){
209         if(maxF>1000000){
210
211             cout << "maxFrequenz: " << maxF/1000000 << " MHz" << endl;
212             }
213             else{
214                 cout << "maxFrequenz: " << maxF/1000 << " kHz" << endl;
215             }
216         }
217
218     cout << "-----" << endl;
219     if(maxF < freq){
220         cout << "Frequenz zu gross" << endl;
221     }
222     else{
223         cout << "Frequenz okay" << endl;
224     }
225 }
226
227
228 }

```

## A.24 cross-compatibility.h

```
1 // Funktionen um Windows und Linux Kompatibilität zu ermöglichen
2
3 #ifndef CLEARSCREEN_H
4 #define CLEARSCREEN_H
5
6 #include <cstdlib>
7 #include <iostream>
8 #include <stdio.h>
9 #include <stdlib.h>
10 #include <string>
11 using namespace std;
12
13 void clear_screen();
14
15 //void debug_msg(char text);
16
17 #endif
18
19 /// Debug Output
20 #if defined DEBUG
21     #define debug_msg(text); cout << text << endl;
22 #else //Assume Release
23     #define debug_msg(text);
24 #endif
25
26
27 /// Debug Pause um Debug Output lesen zu können
28 #if defined DEBUG
29     #define debug_pause(); cin.ignore(); cin.get();
30 #else //Assume Release
31     #define debug_pause();
32 #endif
33
34
35 #if defined _WIN32 || defined _WIN64
36     #define linuxzusatz 0
37 #else
38     #define linuxzusatz 1
39 #endif
```

## A.25 cross-compatibility.cpp

```
1 #include "cross-compatibility.h"
2 using namespace std;
3
4 void clear_screen() {
5     #if defined _WIN32 || defined _WIN64
6         std::system ( "CLS" );
7     #else
8         // Assume POSIX
9         std::system ( "clear" );
10 #endif
11 }
```