

Dokumentation zum Projektpraktikum Informationstechnik

Gruppe: 064
Gruppenmitglieder: Benedikt Braunger, Cornelius Richt,
Fabian Schackmar, Lukas Mohrbacher
Tutor: Florian Brauchle
Abgabetermin: 18.01.2013
Semester: WS2011/2012

Inhaltsverzeichnis

0.1	Aufgabenstellung	3
0.1.1	Zeitplanung	3
0.2	das Programm	3
0.2.1	Menü	3
0.2.2	Bibliothek	5
0.2.3	Gattertyp und Flipflop	10
0.2.4	Grapherzeuger	10
0.2.5	SignallisteErzeuger	16
0.2.6	signal	20
0.2.7	Faktoren	20
0.2.8	LaufzeitAnalysator	20
0.3	Abschlusset	20
0.4	Materialien	22
0.4.1	Software	22
0.4.2	Hilfsmittel	22
0.5	Quellcode	23
A	Quelltext	24
A.1	main.cpp	24
A.2	Menue.h	25
A.3	Menue.cpp	26
A.4	Faktoren.h	32
A.5	Faktoren.cpp	34
A.6	Bibliothek.h	38
A.7	Bibliothek.cpp	39
A.8	GatterTyp.h	46
A.9	GatterTyp.cpp	47
A.10	Flipflop.h	49
A.11	Flipflop.cpp	50
A.12	SignalListeErzeuger.h	51
A.13	SignalListeErzeuger.cpp	52
A.14	signals.h	57
A.15	signals.cpp	58
A.16	SchaltwerkElement.h	60
A.17	SchaltwerkElement.cpp	62
A.18	ListenElement.h	64
A.19	ListenElement.cpp	65
A.20	GraphErzeuger.h	66
A.21	GraphErzeuger.cpp	68
A.22	LaufzeitAnalysator.h	75
A.23	LaufzeitAnalysator.cpp	76
A.24	cross-compatibility.h	81
A.25	cross-compatibility.cpp	82

0.1 Aufgabenstellung

Wie Sie in der Einführung zum Projektpraktikum gelernt haben, weist jedes reale Bauteil 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 -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 Ihres Programms. Dazu muss das Programm den längsten Signalpfad finden und dessen Laufzeit berechnen. Die Laufzeit hängt noch von diversen Äußerer Einflüssen wie Temperatur und Spannung ab, die Sie ebenfalls berücksichtigen sollen.

Dieser stellt im Projektpraktikum das Beispiel- schaltwerk dar, welches untersucht werden soll.

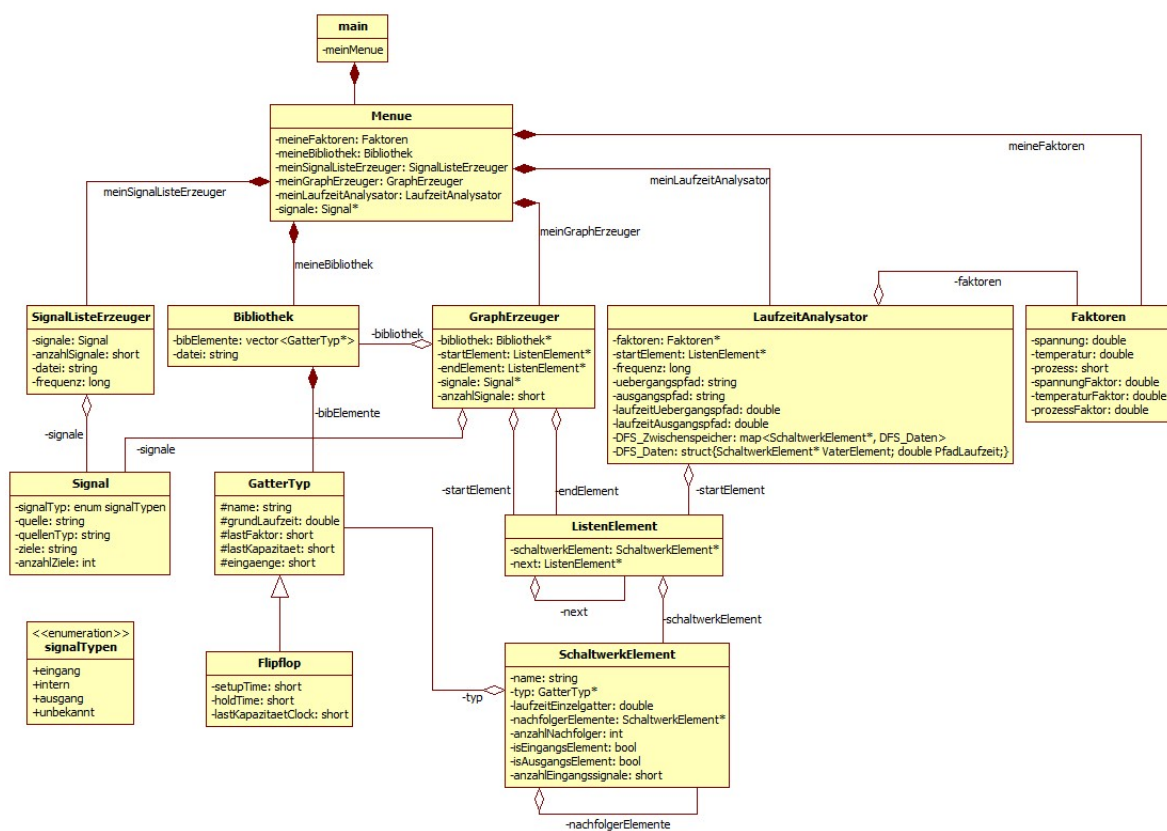


Abbildung 1:

0.1.1 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 zu den Abschlusstest und der Präsentation vor dem Tutor

0.2 das Programm

0.2.1 Menü

Das Menü dient dazu dem Benutzer eine Oberfläche zur Verfügung zu stellen, mit welcher er das Programm möglichst einfach bedienen kann. Dazu gehört die Ausgabe der Ergebnisse so wie die Möglichkeit verschiedene Parameter einzugeben. Zudem werden die einzelnen Instanzen des Programms im Menü verknüpft



Abbildung 2:

0.2.2 Bibliothek

getBibElement(string typ)

Dieser Methode wird ein string, des Gattertyps (z.B. inv1a), übergeben. Sie gibt einen Zeiger auf das entsprechende Element vom Typ GatterTyp zurück.

dateiAusgabe

Ausgabe der Datei auf dem Bildschirm, dabei sollen die Zeilen durchnummeriert werden. Dabei soll, falls die Datei nicht vorhanden ist oder ein Fehler beim Lesen auftritt, das Programm nicht abbrechen, sondern eine Fehlermeldung ausgeben.

```

1 void Bibliothek::dateiAusgabe(void)
2 {
3     ifstream f(datei.c_str());
4
5     string buffer;
6
7     int i=0;
8
9     if (f.good())
10    {
11        while (!f.eof())
12        {
13            getline(f,buffer);
14            cout << i<<": "<<buffer<<endl;
15            i++;
16        }
17    }
18    else
19    {
20        openError();
21    }
22 }
23

```

dateiAuswerten

Die Methode dient zum Einlesen und Auswerten der Bibliotheksdatei. Dabei soll jeder in der Datei beschriebene Gattertyp in einem Element vom Typ GatterTyp im Vektor bibElemente gespeichert werden. Die Reihenfolge ist dabei nicht wichtig. Das Flipflop kann dabei am Namen erkannt werden, welcher als bekannt vorausgesetzt wird. Das Flipflop wird in einem Element vom Typ Flipflop im Vektor bibElemente gespeichert.

Zum Erstellen der FlipFlops einer Kindklasse von GatterTyp haben wir einen zusätzlichen Vector benutzt und diesen nach erfolgreichem Anlegen der Bauteile in den Gattertyp-Vector gecastet

Die Datei wird zeilenweise durchgegangen um die Elemente anzulegen. Ein neues Element beginnt an einer [Klammer und endet an einer leeren Zeile.

```

1 void Bibliothek::dateiAuswerten(void)
2 {
3     ifstream f(datei.c_str());
4
5     string buffer;
6     while (!f.eof())
7     {
8         getline(f,buffer);
9         //"\r" entfernen
10        buffer.erase(buffer.size()-1);
11
12        //von [[Bausteine]] bis Leerzeile einlesen
13        if (buffer.find("[[Bausteine]]")==0)
14        {

```

```

16     while (!f.eof())
17     {
18         getline(f,buffer);

20         if(buffer=="\r")
21         {

23             debug_msg("Blockende gefunden");
24             break;
25         }

27         //"\r" entfernen
28         buffer.erase(buffer.size()-1);


32         /*if(buffer == "dff")
33         {
34             Flipflop* dummy = (new Flipflop());

36             dummy->setName(buffer);

38             bibElemente.push_back(*dummy);

40             debug_msg( "ff angelegt: "<<buffer);

42         }
43         else
44         {
45             GatterTyp* dummy= new GatterTyp();
46             dummy->setName(buffer);

48             bibElemente.push_back(*dummy);
49             debug_msg( "gt angelegt: "<<buffer);

51             }*/

56     }
57 }

59 else if(buffer.find("[")==0)
60 {
61     //Klammern [ ] entfernen
62     string name = buffer.substr(1,buffer.size()-2);


65     //FF anlegen
66     if (name=="dff")
67     {
68         Flipflop *ff = new Flipflop();
69         ff->setName(name);

71         debug_msg(name <<"als FF anlegen");

```

```

74         while (!f.eof())
75         {
76             getline(f,buffer);

78             //Abbruch falls Absatz zu Ende
79             if (buffer=="\r")
80             {

82                 //FF zu bibElemente hinzfügen
83                 bibElemente.push_back((ff));


88                 debug_msg("Ende von: "<<name<<" gefunden");
89                 break;
90             }
91             //"\r" entfernen
92             buffer.erase(buffer.size()-1);


96             //*allgemeine Attribute
97             if (buffer.find("ei:")==0)
98             {
99                 ff->setEingaenge(atoi(buffer.substr(3).c_str()));
100                 debug_msg( "ei init "<<ff->getEingaenge());
101             }

103             else if (buffer.find("cl:")==0)
104             {
105                 ff->setLastKapazitaet(atoi(buffer.substr(3).c_str()));
106                 debug_msg("cl init "<<ff->getLastKapazitaet());
107             }

109             else if (buffer.find("kl:")==0)
110             {
111                 ff->setLastFaktor(atoi(buffer.substr(3).c_str()));
112                 debug_msg("kl init "<<ff->getLastFaktor());
113             }

115             else if (buffer.find("tpd0:")==0)
116             {
117                 ff->setGrundLaufzeit(atof(buffer.substr(5).c_str()));
118                 debug_msg("tpd0 init "<<ff->getGrundLaufzeit());
119             }


122             //*Flipflop Attribute
123             else if (buffer.find("tsetup:")==0)
124             {
125                 ff->setSetupTime(atoi(buffer.substr(7).c_str()));
126                 debug_msg("ff testup init: "<<ff->getSetupTime());

128             }
129             else if (buffer.find("ed:")==0)
130             {
131                 ff->setEingaenge(atoi(buffer.substr(3).c_str()));
132                 debug_msg("ff ed init: "<<ff->getEingaenge());

```

```

134     }
135     else if (buffer.find("thold:") == 0)
136     {
137         ff->setHoldTime(atoi(buffer.substr(6).c_str()));
138         debug_msg("ff thold init: "<<ff->getHoldTime());
139
140     }
141     else if (buffer.find("cd:") == 0)
142     {
143         ff->setLastKapazitaet(atoi(buffer.substr(3).c_str()));
144         debug_msg("ff cd init: "<<ff->getLastKapazitaet());
145
146     }
147     else if (buffer.find("tpdt:") == 0)
148     {
149         ff->setGrundLaufzeit(atof(buffer.substr(5).c_str()));
150         debug_msg("ff tpdt init: "<<ff->getGrundLaufzeit());
151
152     }
153     else if (buffer.find("kl:") == 0)
154     {
155         ff->setLastFaktor(atoi(buffer.substr(3).c_str()));
156         debug_msg("ff kl init: "<<ff->getLastFaktor());
157
158     }
159     else if (buffer.find("ct:") == 0)
160     {
161         ff->setLastKapazitaetClock(atoi(buffer.substr(3).c_str()));
162         debug_msg("ff ct init: "<<ff->getLastKapazitaetClock());
163
164     }
165
166     else
167     {
168         if (buffer.find("#endf") != 0) {
169             // Falls Attribut nicht gefunden
170             readError();
171             debug_msg(buffer<<" nicht gefunden");
172         }
173         else { bibElemente.push_back((ff)); break; }
174     }
175
176 }
177
178 }
179
180 else {
181
182     GatterTyp *gt = new GatterTyp();
183
184     debug_msg(name <<"als GT anlegen");
185     gt->setName(name);
186
187
188     while (!f.eof())
189     {
190
191

```



```

192         getline(f,buffer);

194         //Abbruch falls Absatz zu Ende
195         if (buffer=="\r")
196         {

198             //GT zu bibElemente hinzufügen
199             bibElemente.push_back(gt);


205             debug_msg("Ende von: "<<name<<" gefunden");
206             break;
207         }
208         //" \r" entfernen
209         buffer.erase(buffer.size()-1);


213         ///*allgemeine Attribute
214         if (buffer.find("ei:")==0)
215         {
216             gt->setEingaenge(atoi(buffer.substr(3).c_str()));
217             debug_msg( "ei init " <<gt->getEingaenge());
218         }

220         else if (buffer.find("cl:")==0)
221         {
222             gt->setLastKapazitaet(atoi(buffer.substr(3).c_str()));
223             debug_msg("cl init " <<gt->getLastKapazitaet());
224         }

226         else if (buffer.find("kl:")==0)
227         {
228             gt->setLastFaktor(atoi(buffer.substr(3).c_str()));
229             debug_msg("kl init " <<gt->getLastFaktor());
230         }

232         else if (buffer.find("tpd0:")==0)
233         {
234             gt->setGrundLaufzeit(atof(buffer.substr(5).c_str()));
235             debug_msg("tpd0 init " <<gt->getGrundLaufzeit());
236         }


239         else
240         {
241             if (buffer.find("#endf")!=0){
242                 //Falls Attribut nicht gefunden
243                 readError();
244                 debug_msg(buffer<<" nicht gefunden");
245             }
246             else break;
247         }

249     }

```

```

252         }
254     }
255 }

259     for(int h=0;h<bibElemente.size();h++){
260         debug_msg( bibElemente[h]->getName());
261     }

268 }

```

Fehlerausgabe

Die Fehlerausgabe findet in folgenden Methoden statt. Dies ermöglicht eine bessere Fehlerbehandlung, zB die Ausgabe von verschiedenen Sprachen.

C++ bietet allerdings noch weit aus bessere Möglichkeiten Fehler richtig abzufangen

```

1  /**Ausgabe einer Fehlermeldung beim Ändern einer Datei. */
2  void Bibliothek::openError(void)
3  {
4      cerr <<"OPEN ERROR"<<endl;
5
6  }
7  /**Ausgabe einer Fehlermeldung beim Lesen einer Datei.
8  */
9  void Bibliothek::readError(void)
10 {
11     cerr <<"READ ERROR"<<endl;
12
13 }

```

0.2.3 Gattertyp und Flipflop

Die Klasse Flipflop ist eine Erweiterung der Klasse Gattertyp beide stellen Bauteile da, welche im Schaltwerk verwendet werden. Sie werden in der Bibliothek initialisiert.

0.2.4 Grapherzeuger

Der Grapherzeuger legt eine verkettete Liste aus den Schaltwerkelementen an und erzeugt somit die Schaltung aus den einzelnen Bauteilen. Zudem wird die Schaltung auf offene Eingänge geprüft. Mit der Methode run wird überprüft ob die Eingänge der Schaltung mit den vorhandenen Eingängen übereinstimmt, ist dies nicht der Fall wird ein entsprechender Fehler ausgegeben und die Erzeugung des Graphen abgebrochen.

```

1  startElement = NULL; ///Initialisierung
2      endElement = NULL;
3      signale = NULL;
4      anzahlSignale = 0;
5      gAnzahl = 0;

7      short eingaege = 0;
8      short ausgaenge = 0;

10     ListenElement* tmpElement = NULL;
11     anzahlSignale = signallist.getAnzahlSignale();

```

```

13  /// geht Signalliste durch
14  for (int i = 0; i < anzahlSignale; i++)
15  {
16      Signal tmpSignal = *signallist.getSignal( i );

17      if ((tmpSignal.getSignalTyp() == eingang) )    /// prueft ob Eingang
18      {
19          debug_msg("INFO: eingang gefunden");
20          eingange++;
21      }
22      else if (tmpSignal.getSignalTyp() == clk)    /// prueft ob Takt
23      {
24          debug_msg("INFO: clock gefunden");
25      }
26      else if ((tmpSignal.getSignalTyp() == intern) or (tmpSignal.getSignalTyp() == <-
27      ausgang)) /// wenn intern oder ausgangs-signal hat das signal eine quelle<-
28      {
29          /// die man in Schaltwerke ueberfuehren kann
30          if ( tmpSignal.getQuelle() != "" )    /// zum abfangen von unbenutzten <-
31          Signalen
32          {
33              GatterTyp* tmpGatter = bibliothek->getBibElement(tmpSignal.<-
34                  getQuellenTyp()); // kann man sich theoretisch auch sparen und <-
35                  alle tmpGatter durch bibliothek->getBibElement(tmpSignal.<-
36                  getQuellenTyp()) ersetzen

37              ListenElement* newListElement = new ListenElement();
38              SchaltwerkElement* newSchaltwerkElement = new SchaltwerkElement( <-
39                  tmpGatter );

40              /// Schaltwerk uebernimmt Daten des Signals
41              newSchaltwerkElement->setName(tmpSignal.getQuelle());
42              newSchaltwerkElement->setAnzahlNachfolger(tmpSignal.getAnzahlZiele())<-
43              ;
44              newSchaltwerkElement->setLaufzeitEinzelgatter( tmpGatter-><-
45                  getGrundlaufzeit() );
46              newSchaltwerkElement->setAnzahlEingangssignale( tmpGatter-><-
47                  getEingaenge());

48              /// pruefen ob Ausgang
49              if ( tmpSignal.getSignalTyp() == ausgang )
50              {
51                  newSchaltwerkElement->setIsAusgangsElement(true);
52                  debug_msg("INFO: ausgang gefunden");
53                  ausgaenge++;
54              }

55              /// verknuepfen von Schaltwerkselement mit Listenelement
56              newListElement->setSchaltwerkElement( newSchaltwerkElement );
57              gAnzahl++;
58              debug_msg("INFO: "<<gAnzahl<<" . ListenElement angelegt vom Typ "<< <-
59                  tmpSignal.getQuellenTyp()<<" !");

60              /// baut die Liste auf
61              if ( startElement == NULL ) /// ist nur NULL, wenn noch kein Element <-
62              der Liste existiert

```

```

59         {
60             endElement = newListElement;
61             startElement = newListElement;
62
63         }
64         else
65         {
66             tmpElement->setNextElement( newListElement );
67             endElement = newListElement;
68         }
69         tmpElement = newListElement;
70
71     }
72     else // von leerer Quelle Abfrage, um ungenutzte Signale zu erkennen
73     {
74         cout << "Fehler! Unbenutztes Signal gefunden" << endl;
75         cin.ignore();
76         cin.get();
77     }
78 }
79
80 else // von Signaltypabfrage
81 {
82     cout << "Fehler! Unbekannter Signaltyp" << endl;
83     cin.ignore();
84     cin.get();
85 }
86 }
87 // eingang finden
88 for (int z = 0; z < signallist.getAnzahlSignale(); z++) // geht die ←
89     Sigalliste durch
90 {
91     if ( signallist.getSignal(z)->getSignalTyp() == eingang) // vergleicht ←
92         mit Signaltypen, ob "eingang" der Signaltyp ist
93     {
94         debug_msg( "INFO: Dieses Eingangssignal hat "<<signallist.getSignal(z)->←
95             getAnzahlZiele()<< " Ziel(e)");
96         for ( int y = 0; y < signallist.getSignal(z)->getAnzahlZiele(); y++) ←
97             // durchläuft alle ziele dieses signals
98         {
99             string eingangsGatter = signallist.getSignal(z)->getZiel( y );
100             for (ListenElement* ptr = startElement; ptr != NULL; ptr = ptr->←
101                 getNextElement()) // und gleicht die ziele mit den ←
102                 schaltwerksnamen in dem
103             {
104                 if ( ptr->getSchaltwerkElement()->getName() == eingangsGatter ) ←
105                     // jeweiligen listenelement ab
106                 {
107                     ptr->getSchaltwerkElement()->setIsEingangsElement(true);
108                     debug_msg( "INFO: "<< ptr->getSchaltwerkElement()->getName() ←
109                         <<" ist Eingang");
110                 }
111             }
112         }
113     }
114 }
115
116 // prueft, ob es unbeschaltete Eingänge gibt
117 short tmpZaehler ;

```

```

110     for (ListenElement* ptr = startElement; ptr != NULL; ptr = ptr->getNextElement()↵
111         ) /// durchlaeuft die Listenelemente
112     {
113         tmpZaehler = 0;
114
115         debug_msg("INFO:-----");
116
117         for (int z = 0; z < signallist.getAnzahlSignale(); z++) ///danach die ↵
118             Signalliste
119         {
120             for (int r = 0; r < signallist.getSignal(z)->getAnzahlZiele(); r++) /// ↵
121                 und die Ziele eines jeden Signals
122             {
123                 /// prueft, ob das Signalziel mit dem SchaltwerkElementsnamen ↵
124                 uebereinstimmt und erhoeht den Eingangszaehler bei Erfolg um 1
125                 if ( signallist.getSignal(z)->getZiel(r) == ptr->getSchaltwerkElement↵
126                     ()->getName())
127                 {
128                     tmpZaehler += 1;
129
130                     debug_msg("INFO: "<< tmpZaehler <<". Eingang von "<< ptr->↵
131                         getSchaltwerkElement()->getName() <<" gefunden");
132                 }
133             }
134         }
135         /// falls dff mit clk, hat die Bib einen Eingang zu wenig, wird hier ↵
136         korrigiert
137         if (ptr->getSchaltwerkElement()->getTyp()->getName()=="dff") // weil der ↵
138             clock eingang nicht mit eingelesen wird.. sollte man vlt noch aendern
139         {
140             tmpZaehler -= 1;
141         }
142         /// check, ob Schaltwerk und Bib fuer das jeweilige Element dieselbe Anzahl ↵
143         Eingaenge verzeichnet haben
144         if (ptr->getSchaltwerkElement()->getTyp()->getEingaenge() != tmpZaehler)
145         {
146             cout << "Fehler!\nAnzahl Eingaenge laut Bibliothek: \t"<<ptr->↵
147                 getSchaltwerkElement()->getTyp()->getEingaenge()<<endl
148             <<"Anzahl Eingaenge laut Schaltwerk: \t"<<tmpZaehler << endl;
149             cin.ignore();
150             cin.clear();
151             cin.get();
152         }
153     }
154 }
155
156 void GraphErzeuger::graphErzeugen(SignallisteErzeuger signallist)
157 {
158     /// durchlaeuft die Liste der durch ListenElemente verknuepften Schaltwerke
159     for (ListenElement* ptr = startElement; ptr != NULL; ptr = ptr->getNextElement()↵
160         )
161     {
162         ListenElement* tmpListenElement = ptr;
163         SchaltwerkElement* tmpSWE = tmpListenElement->getSchaltwerkElement();
164
165         /// prueft ob ein Schaltwerk maximal 5 Nachfolger besitzt

```

```

158     if ( tmpSWE->getAnzahlNachfolger() <= 5)
159     {

161         /// durchlaeuft die Signalliste auf der Suche nach gleichnamigen Quellen ←
            der Signale und Schaltwerksnamen
162         for (int i = 0; i < signallist.getAnzahlSignale(); i++)
163         {

165             Signal tmpSignal = *signallist.getSignal( i );

167             if ( tmpSignal.getQuelle() == tmpSWE->getName())
168             {

170                 /// bei Treffer wird wieder die Signalliste durchlaufen auf der ←
                    Suche nach den Zielen des gleichnamigen Signals
171                 for ( int j = 0; j < tmpSignal.getAnzahlZiele(); j++)
172                 {

173                     string folgeGatter = tmpSignal.getZiel( j );

175                     /// sucht zu den Zielen des Signals das entsprechende ←
                        Schaltwerk aus den ListenElementen
176                     for (ListenElement* ptr2 = startElement; ptr2 != NULL; ptr2 = ←
                        ptr2->getNextElement())
177                     {

178                         ListenElement* tmpListenElement2 = ptr2;
179                         if ( tmpListenElement2->getSchaltwerkElement()->getName() ←
                            == folgeGatter )
180                         {

181                             tmpListenElement->getSchaltwerkElement()->←
                                nachfolgerHinzufuegen( tmpListenElement2->←
                                    getSchaltwerkElement(), j );
182                             debug_msg( "INFO: "<< tmpListenElement2->←
                                    getSchaltwerkElement()->getName() << " ist ←
                                        Nachfolger von " << tmpListenElement->←
                                            getSchaltwerkElement()->getName());
183                         }
184                     }
185                 }

187             }

189         }

191     }
192     else        /// von AnzahlNachfolger if-Abfrage
193     {

194         cout << "Fehler: Mehr als 5 Nachfolgegatter bei "<< tmpSWE->getName() << ←
            endl;
195     }

197 }

199 }

201 void GraphErzeuger::listenAusgabe ( )        /// gibt die Listenelemente mit ←
            Gatternamen und ihre Nachfolgegatter aus
202 {

204     for ( ListenElement* ptr = startElement; ptr != NULL; ptr = ptr->getNextElement()←

```

```

205 {
206     () // geht die ListenElemente durch
207
208     cout << "-----\n"<<endl
209     <<"Gattername: \t\t" << ptr->getSchaltwerkElement()->getName() <<endl <←
210         // Gattername
211     <<"Gattertyp: \t\t" << ptr->getSchaltwerkElement()->getTyp()->getName() <<←
212         endl; // GatterTyp
213
214     // evtl. zusaetzliche Ausgaben wie "Eingang", "Ausgang", "←
215     // LaufzeitEinzelGatter", fuer FlipFlops noch andere Attribute
216     if ( ptr->getSchaltwerkElement()->getIsEingangselement() == true) // kann ←
217         // man sich mal noch ueberlegen in die Ausgabe mit aufzunehmen
218     {
219         cout <<"Schaltungseingangselement"<<endl;
220     }
221     if ( ptr->getSchaltwerkElement()->getIsAusgangselement() == true)
222     {
223         cout<<"Schaltungsausgangselement"<< endl;
224     }
225     cout<<"Laufzeit Einzelgatter: \t"<< ptr->getSchaltwerkElement()->←
226         getLaufzeitEinzelgatter() <<endl;
227
228     cout << "Is Flipflop: \t\t"<< (( Flipflop*) (ptr->getSchaltwerkElement()->←
229         ->getTyp()) )->getIsFlipflop()<<endl;
230
231     if (ptr->getSchaltwerkElement()->getTyp()->getName()=="dff")
232     {
233         cout << "Setup-Time: \t\t" << (( Flipflop*) (ptr->getSchaltwerkElement()->←
234             getTyp()) )->getSetupTime() << endl
235         << "Hold-Time \t\t" << (( Flipflop*) (ptr->getSchaltwerkElement()->getTyp()←
236             )) )->getHoldTime()<<endl
237         << "Lastkapazitaet: \t"<< (( Flipflop*) (ptr->getSchaltwerkElement()->←
238             getTyp()) )->getLastKapazitaetClock()<<endl;
239     }
240
241     cout<<"Anzahl Eingangssignale: "<< ptr->getSchaltwerkElement()->←
242         getAnzahlEingangssignale() <<endl; // Ende Fakultative Ausgabe
243
244     // Ausgabe der Anzahl der Folgegatter und dann der Gatter mit ihrem Namen
245     if (ptr->getSchaltwerkElement()->getAnzahlNachfolger()==1){
246         cout<<"-->Das Gatter hat "<< ptr->getSchaltwerkElement()->←
247             getAnzahlNachfolger()<<" Ziele" <<endl; // Einzahl
248     }else if (ptr->getSchaltwerkElement()->getAnzahlNachfolger()==0){
249         cout<<"-->Das Gatter hat keine Ziele" <<endl; <←
250             // Keine
251     }else{
252         cout<<"-->Das Gatter hat "<< ptr->getSchaltwerkElement()->←
253             getAnzahlNachfolger()<<" Ziele" <<endl; // Mehrzahl
254     }
255
256     if (ptr->getSchaltwerkElement()->getAnzahlNachfolger()!=0) // falls ←
257         // Nachfolger existieren
258     {
259
260         string ausgabe = " ";
261         for ( int s = 0; s < ptr->getSchaltwerkElement()->getAnzahlNachfolger() ; <←
262             s++) // werden alle Nachfolgernamen in einen Ausgabe string ←
263             // geschrieben

```

```

247     {
248
249         ausgabe = ausgabe + ptr->getSchaltwerkElement()->getNachfolger(s)->
            getName() + " ";
250     }
251     cout << "Angeschlossene Gatter:\t"<<ausgabe <<endl;
252 }
253 else /*if (ptr->getSchaltwerkElement()->getAnzahlNachfolger()==0)*/ /// falls
    keine Nachfolger existieren
254 {
255     cout << ptr->getSchaltwerkElement()->getName() << " hat keine Folgegatter"
        "<<endl;
256 }
257 }
258 }
259 }

```

0.2.5 SignallisteErzeuger

SignallisteErzeuger liest die gegebene Datei ähnlich wie in der Bibliothek ein und speichert diese in einem Vector

```

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 Menü¼ aufrufen
14 }
15
16 /**Destruktor
17 */
18 SignallisteErzeuger::~SignallisteErzeuger()
19 {
20     //dtor
21 }
22
23 /**dateiAusgabe
24 Öffnet die Datei die in der 'datei' Variable der Klasse gespeichert ist und gibt
    die aus
25 */
26 void SignallisteErzeuger::dateiAusgabe(void)
27 {
28     ifstream f(datei.c_str());
29
30     string buffer;
31
32     int i=0;
33
34     if(f.good())
35     {
36         while (!f.eof())
37         {

```



```

38         getline(f,buffer);
39         cout << i<<": " <<buffer << endl;
40         i++;
41     }
42 }
43 else
44 {
45     cout << "ERR: Can not read file!";
46 }
47 }
48
49 /** Gibt Signal aus dem 'signale' Vektor an der im Parameter spezifizierten Stelle ←
    zurÄ½ck
50 */
51 Signal* SignallisteErzeuger::getSignal(int i) {
52     return &signale.at(i) ;
53 }
54
55 /**Liest die 'datei' aus und beginnt mit der Auswertung
56 */
57 int SignallisteErzeuger::readFile() {
58     signale.clear();                                     //Vektor 'signale' wird geleert
59     string line;
60     ifstream listfile(getDatei().data());               //Ä½ffne Dateistream
61     Signal* bufferobj = new Signal;
62     signale.push_back( *bufferobj );                    //Reserviere leeres Objekt fÄ½r die ←
        CLOCK
63     if (listfile.is_open()) {
64         //debug_msg( "INFO: file is open" );
65         while (!listfile.eof()) {
66             getline(listfile,line);                     //liest Zeile fÄ½r Zeile aus
67             if (((line.substr(0,2)) == "//") or (line == "\r") or (line == "")) {
68                 debug_msg( "INFO: drop, comment or empty line" );
69             }else if ((line.substr(0,12)) == "ARCHITECTURE") { ←
                //Wenn Kommentar, leere Zeile oder Schwachsinn drin steht, passiert ←
                gar nichts
70                 debug_msg( "INFO: drop, ARCHITECTURE shit" );
71             }else if ((line.substr(0,6)) == "ENTITY") {
72                 while (1) {
73                     getline(listfile,line);
74                     if ((line.substr(0,2)) == "//") {
75                         debug_msg( "INFO: drop, comment or empty line" );
76                     }else if ((line.substr(0,5)) == "INPUT") {
77                         debug_msg( "INFO: Found INPUT line!" );
78                         readSignalLine(eingang,5,line);
79                     }else if ((line.substr(0,6)) == "OUTPUT") {
80                         debug_msg( "INFO: Found OUTPUT line!" );
81                         readSignalLine(ausgang,6,line);
82                     }else if ((line.substr(0,7)) == "SIGNALS") {
83                         debug_msg( "INFO: Found SIGNALS line!" );
84                         readSignalLine(intern,7,line);
85                     }else if ((line.substr(0,5)) == "CLOCK") {
86                         debug_msg( "INFO: Found CLOCK line!" );
87                         string hr_frequency = line.substr(11,(line.length()-11)); ←
                            //Schneide Frequenz aus
88                         frequenz = atoi(hr_frequency.data()); ←
                            //Lese Frequenzzahl
89                         if (hr_frequency.substr(hr_frequency.size()-5,1)=="M") { ←
                            //Multipliziere ←

```

```

    frequenz
    frequenz = frequenz * 1000000;
} else if (hr_frequency.substr(hr_frequency.size()-5,1)=="k") {
    frequenz = frequenz * 1000;
}
bufferobj->setSignalTyp(clk);
signale.at(0) = *bufferobj;
debug_msg( "INFO: Set clk to: " << frequenz );
} else if (line == "\r" or (line == "")){
    debug_msg( "INFO: Found empty line, leave ENTITY area!" );
    break;
} else {
    debug_msg( "ERR: Error reading line" );
    break;
}
}
} else if ((line.substr(0,5)) == "BEGIN") {
    while (1) {
        getline(listfile,line);
        if ((line.substr(0,2)) == "//") {
            debug_msg( "comment" );
        } else if ((line.substr(0,1)) == "g") {
            debug_msg( "INFO: Found GATE line!" );
            if (readGateLine(line) == 1 ) { ←
                //Wenn Kurzschluss ←
                bereits vorhanden
                cout << "ERR: Short curcuit" << endl;
                cin.get();
                return 21;
            }
        } else if ((line.substr(0,6)) == "END") {
            debug_msg( "INFO: Found END line!" );
            signalTypen tmpsig;
            tmpsig = signale.at(0).getSignalTyp();
            debug_msg( "DEBUG "<< tmpsig );
            setAnzahlSignale(signale.size()); ←
                //AnzahlSignale auf die←
                Größe des Vektor setzen
            debug_msg( "DEBUG: AnzahlSignale: " << getAnzahlSignale() )←
                ;
            return 0;
        } else {
            debug_msg( "ERR: Error reading line" );
            break;
        }
    }
} else { //-----else
    debug_msg( "ERR: Error reading headline" );
    break;
}
}
} else {
    cout << "ERR: Error opening file!";
    cin.get();
    return 1;
}
return 0;
}
}

```

```

143 int SignallisteErzeuger::readSignalLine(signalTypen typ, int lengthBegin, string <-
    tmpLine) {
144     string tmpSignal;
145     stringstream tmpStream(tmpLine.substr(lengthBegin+1,(tmpLine.length()-(<-
        lengthBegin+2+linuxzusatz)))); ///Erstellt 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 }

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 Gatetype <-
        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 für Ausgangssignal
166     }
167     else {
168         return 1;
169     }
170     signale.at(atoi(tmpOut.c_str())).setQuellentyp(gatetype); <-
        ///Setze Quellentyp für 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) <-

```

```

181         ;          //Füge Ziele zu aktuellem Signal hinzu
182     }
183     return 0;
184 }

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 }

```

0.2.6 signal

Ist eine Klasse um Signalattribute zu speichern

0.2.7 Faktoren

Speichert Attribute und berechnet Faktoren die sich auf die Laufzeit auswirken, wie zB Temperatur und Herstellungsprozesse. Zudem ist die Klasse für die Interpolation verantwortlich

0.2.8 LaufzeitAnalysator

berechne LaufzeitEinzelgatter

In dieser Methode wird die Laufzeit der einzelnen Gatter abhängig von äußeren Faktoren berechnet

Zyklensuchse

Die Zyklensuche prüft die Schaltung auf einen eventuell vorhandenen Zyklus, damit die Laufzeitanalyse darauf reagieren kann

LaufzeitAnalysator DFS Visit

Die Tiefensuche durchsucht die einzelnen Äste des Baumes nach evtuellen Zyklen und berechnet den längsten Übergangspfad sowie Ausgangspfad

maxFrequenz

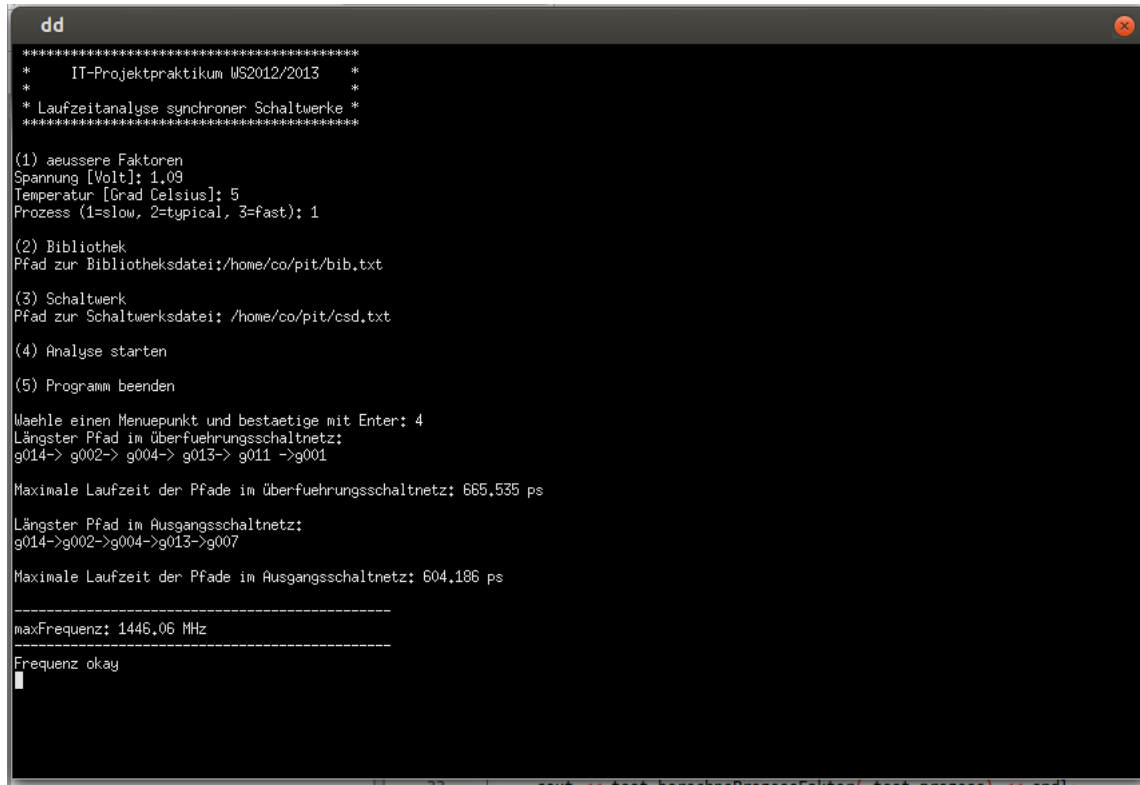
Im Folgenden wird die maximale Frequenz berechnet, mit welcher die Schaltung betrieben werden kann und geprüft ob die vorgesehene Frequenz die maximale Frequenz nicht überschreitet

0.3 Abschlussest

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



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

(1) aeußere Faktoren
Spannung [Volt]: 1.09
Temperatur [Grad Celsius]: 5
Prozess (1=slow, 2=typical, 3=fast): 1

(2) Bibliothek
Pfad zur Bibliotheksdatei: /home/co/pit/bib.txt

(3) Schaltwerk
Pfad zur Schaltwerksdatei: /home/co/pit/csd.txt

(4) Analyse starten

(5) Programm beenden

Wähle einen Menüpunkt und bestätige mit Enter: 4
Längster Pfad im Überführungsschaltnetz:
g014-> g002-> g004-> g013-> g011 ->g001

Maximale Laufzeit der Pfade im Überführungsschaltnetz: 665.535 ps

Längster Pfad im Ausgangsschaltnetz:
g014->g002->g004->g013->g007

Maximale Laufzeit der Pfade im Ausgangsschaltnetz: 604.186 ps

-----
maxFrequenz: 1445.06 MHz
-----
Frequenz okay
█
```

Abbildung 3:

0.4 Materialien

0.4.1 Software

Codeblocks Linux (mit cross compatibility zu Windows) Wir entwickelten unser Programm unter Linux und passten es entsprechend an, damit es auch unter Windows ausführbar ist

0.4.2 Hilfsmittel

Als Hilfsmittel benutzen wir das C++ Kompendium

0.5 Quellcode

Anhang A

Quelltext

A.1 main.cpp

```
1 #include <iostream>
2 #include "Menue.h"

6 using namespace std;

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"
12 using namespace std;
14 class Menue
15 {
16     public:
17         Menue();
19         virtual ~Menue();
21         void start();
23     protected:
26     private:
28         Faktoren meineFaktoren;
29         Bibliothek meineBibliothek;
30         SignallisteErzeuger meinSignallisteErzeuger;
32         GraphErzeuger meinGraphErzeuger;
36         void faktorenMenue();
38         void bibliothekMenue();
40         void schaltwerkMenue();
42         void analyse();
44         void menueKopf();
46         string input;
48 };
50 #endif // MENUE_H
```

A.3 Menue.cpp

```

1 #include "Menue.h"

5 /**
6  Menue Klasse
7  Zustandig für Ein/Ausgabe und Navigation durch das Programm

9  Eine Menüführung innerhalb der Konsole lässt sich am einfachsten realisieren, ←
    indem man zunächst den
10 Inhalt der Konsole löscht, 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"); löscht den Inhalt der Konsole.

16 Erwünschte Ausgabe:
17 *****
18 *      IT-Projektpraktikum WS2012/2013 *
19 *                                     *
20 * Laufzeitanalyse synchroner Schaltwerke *
21 *****

23 (1) äussere Faktoren
24 Spannung [Volt]: 1.2
25 Temperatur [Grad Celsius]: 55
26 Prozess (1=slow, 2=typical, 3=fast): 1

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

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

34 (4) Analyse starten

36 (5) Programm beenden

38 Wähle einen Menüpunkt und bestätige mit Enter:
39 */

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;

```

```

55 // GraphErzeuger meinGraphErzeuger = new GraphErzeuger;
56 // LaufzeitAnalysator meinLaufzeitAnalysator = new LaufzeitAnalysator;
57 // Signal* signale = new Signal;
58 }

60 Menue::~Menue()
61 {
62     /**
63     ist der Destruktor der Klasse.
64     */
65 }

67 void Menue::start()
68 {
69     /**
70     schreibt das Hauptmenü in die Konsole und startet die Hauptschleife, in der
71     durch das Hauptmenü
72     navigiert wird.
73     */

74     while(input != "5") {
75         menueKopf();

76
77         /// Faktoren Hauptmenüpunkt
78         cout << "(1) aeussere Faktoren \nSpannung [Volt]: " << meineFaktoren.
79         getSpannung() << endl;
80         cout << "Temperatur [Grad Celsius]: " << meineFaktoren.getTemp() << endl;
81         cout << "Prozess (1=slow, 2=typical, 3=fast): " << meineFaktoren.getProzess()
82         << endl;

83         cout << endl;

84         /// Bibliothek Hauptmenüpunkt
85         cout << "(2) Bibliothek" << endl;
86         cout << "Pfad zur Bibliotheksdatei:" << meineBibliothek.getPfad() << endl;

87         cout << endl;

88         /// Schaltwerk Hauptmenüpunkt
89         cout << "(3) Schaltwerk \nPfad zur Schaltwerksdatei: " <<
90         meinSignalListeErzeuger.getDatei() << endl;

91         cout << "\n(4) Analyse starten \n\n(5) Programm beenden\n\nWaehle einen
92         Menuepunkt und bestaetige mit Enter: ";

93
94         getline(cin, input);
95         switch (atoi(input.c_str()))
96         {
97             case 1:
98                 faktorenMenue();
99                 break;
100             case 2:
101                 bibliothekMenue();
102                 break;
103             case 3:
104                 schaltwerkMenue();
105                 break;
106             case 4:

```

```

109     analyse();
110     break;
111 }
112 }
113 }

115 void Menue::faktorenMenue()
116 {
117     /**
118      * Im Untermenü der Äusseren Faktoren sollen die Aussenbedingungen geändert und
119      * die daraus resultierenden Faktoren ausgegeben werden können. Dazu wird von der Klasse Faktoren
120      * eine Ausgabemethode bereitgestellt.
121      */
122     while(input != "5") {
123         menueKopf();
124         cout << "Untermenue Aeussere Faktoren" << endl;
125         cout << "(1) Spannung [Volt]: " << meineFaktoren.getSpannung() << endl;
126         cout << "(2) Temperatur [Grad Celsius]: " << meineFaktoren.getTemp() << endl;
127         cout << "(3) Prozess (1=slow, 2=typical, 3=fast): " << meineFaktoren.getProzess() << endl;
128         cout << "(4) Ausgabe errechneter Faktoren" << endl;
129         cout << "(5) Hauptmenue" << endl << endl;
130         cout << "Waehle einen Menuepunkt und bestaetige mit Enter: ";

132         getline(cin, input);
133         switch (atoi(input.c_str())) {
134             case 1:
135                 cout << "Neue Spannung eingeben: ";
136                 getline(cin, input);
137                 if ( (atof(input.c_str()) >= 1.08) && (atof(input.c_str()) <= 1.32) ) {
138                     meineFaktoren.setSpannung(atof(input.c_str()));
139                 } else {
140                     cout << "Die Spannung muss zwischen 1.08 und 1.32 liegen" << endl;
141                     cin.get();
142                 }
143                 break;
144             case 2:
145                 cout << "Neue Temperatur eingeben: ";
146                 getline(cin, input);
147                 if ( (atof(input.c_str()) >= -25) && (atof(input.c_str()) <= 125) ) {
148                     meineFaktoren.setTemp(atof(input.c_str()));
149                 } else {
150                     cout << "Die Temperatur muss zwischen -25 und 125 liegen!" << endl;
151                     cin.get();
152                 }
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 }

173 void Menue::bibliothekMenue()
174 {
175     /**
176      Im Untermenü der Bibliothek soll der Pfad zur Bibliotheksdatei geändert werden können und man
177      soll sich zur Kontrolle auch die Datei im Menü anzeigen lassen können. Auch die Klasse Bibliothek
178      stellt dazu eine Ausgabemethode bereit.
179     */

181     string pf;
182     while(input != "3") {
183         menueKopf();
184         cout << "Untermenue Bibliothek" << endl;
185         cout << "(1) Pfad zur Bibliotheksdatei: " << meineBibliothek.getPfad() << endl;
186         ;
187         cout << "(2) Ausgabe der Bibliotheksdatei" << endl;
188         cout << "(3) Hauptmenue" << endl << endl;
189         cout << "Waehle einen Menuepunkt und bestaetige mit Enter: ";

190         getline(cin, input);
191         switch (atoi(input.c_str())) {
192             case 1:
193                 cout << "Pfad eingeben: ";
194                 cin >> pf;
195                 if (!meineBibliothek.pfadEinlesen(pf)) {
196                     cout << "Fehler beim einlesen!" << endl;
197                     cin.get();
198                 } else {
199                     meineBibliothek.dateiAuswerten();
200                     meinGraphErzeuger.setBibliothek(&meineBibliothek);
201                 }
202                 break;
203             case 2:
204                 meineBibliothek.dateiAusgabe();
205                 cin.get();
206                 break;
207         }
208     }
209     input.clear();
210 }

212 void Menue::schaltwerkMenue()
213 {
214     /**
215      Im Untermenü des Schaltwerks soll der Pfad zur Schaltwerksdatei veränderbar sein. Zur Kon-
216      trolle soll diese ausgegeben werden können. Ausserdem soll eine Liste der Signale und die Gra-
217      phstruktur ausgegeben werden können. Zu diesen Ausgaben werden Methoden durch die Klassen

```

```

218 SignalListeErzeuger und LaufzeitAnalysator bereitgestellt.
219 */
220 while(input != "5") {
221     string pf;
222     menueKopf();
223     cout << "Untermenue Schaltwerk" << endl;
224     cout << "(1) Pfad zur Schaltnetzdatei: " << meinSignalListeErzeuger.getDatei()
225         << endl;
226     cout << "(2) Ausgabe der Schaltnetzdatei" << endl;
227     cout << "(3) Ausgabe der Signale" << endl;
228     cout << "(4) Ausgabe der Graphstruktur" << endl;
229     cout << "(5) Hauptmenue\n\nWaehle einen Menuepunkt und bestaetige mit Enter: ";
230
231     getline(cin, input);
232     switch (atoi(input.c_str())) {
233     case 1:
234         if (meineBibliothek.getPfad() != "") {
235             cout << "Pfad eingeben: ";
236             cin >> pf;
237             ifstream f(pf.c_str());
238             if (!f.good()) {
239                 cout << "Datei nicht gefunden!" << endl;
240                 cin.ignore();
241                 cin.get();
242             } else {
243                 meinSignalListeErzeuger.setDatei(pf);
244                 if (meinSignalListeErzeuger.readFile() == 21) {
245                     //cout << "Kurzschluss gefunden!" << endl;
246                     cin.get();
247                 } else {
248                     debug_pause();
249                     meinGraphErzeuger.listeAnlegen(meinSignalListeErzeuger);
250                     meinGraphErzeuger.graphErzeugen(meinSignalListeErzeuger);
251                 }
252                 debug_pause();
253             }
254         } else {
255             cout << "\n Die Bibliothek muss als erstes geladen werden!";
256             cin.get();
257         }
258         break;
259     case 2:
260         meinSignalListeErzeuger.dateiAusgabe();
261         cin.get();
262         break;
263     case 3:
264         cout << "Vector size: " << meinSignalListeErzeuger.getAnzahlSignale() << endl;
265         cout << "Vectorcontent:" << endl;
266         for (int i=0; i<meinSignalListeErzeuger.getAnzahlSignale(); i++) {
267             cout << "-----\n";
268             cout << "Nummer: " << i << endl;
269             cout << "Signaltyp: " << meinSignalListeErzeuger.getSignal(i)->getSignalTyp() << endl;
270             cout << "Quelle: " << meinSignalListeErzeuger.getSignal(i)->getQuelle()
271                 << 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 }

289 void Menue::analyse()
290 {
291     /**
292     ruft die zur Analyse benoetigten 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();

298         if (lza.DFS_startSuche(&meinGraphErzeuger)){

300             lza.maxFrequenz(meinSignalListeErzeuger.getFrequenz());
301         }
302     } else {
303         cout << "Es sind noch nicht alle benoetigten Parameter ausgefüllt!";
304     }
305     cin.get();
306     input.clear();
307 }

309 void Menue::menueKopf()
310 {
311     /**
312     Gibt den Kopf der Menüs aus. Dieser bleibt in Hauptmenüs und allen Untermenüs gleich.
313     */
314     clear_screen();
315     cout << " ***** \n * IT-Projektpraktikum \n * \n * Laufzeitanalyse synchroner Schaltwerke * \n *****" << endl << endl; //Ausgabe des "Headers"
316 }

```

A.4 Faktoren.h

```

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

```



```

41  double berechneFaktor (double wert, double arr[][2], int laenge);

43  /** Diese Methode interpoliert einen Wert zwischen zwei vorgegebenen Punkten im 2D-
44  -Raum. Dabei bestimmen x1,
45  y1 und x2, y2 jeweils die Koordinaten der zwei Punkte zwischen denen interpoliert
46  werden soll. Der
47  Ã¼bergabeparameter wert bestimmt den x-Wert des gesuchten Wertes, dabei gilt x1 <
48  wert < x2.*/
49  double interpolation ( double wert, double x1, double x2, double y1, double y2);

50  public:
51
52  /** Konstruktor der Klasse. Er soll beim Anlegen der Klasse alle Attribute mit dem
53  Wert 0 initialisieren.*/
54  Faktoren();
55
56  /** Destruktor der Klasse. */
57  ~Faktoren();
58
59  /** Diese Methode dient zum Lesen des privaten Attributes spannung */
60  double getSpannung();
61
62  /** Diese Methode dient zum Lesen des privaten Attributes temp (die Temperatur) */
63  double getTemp();
64
65  /** Diese Methode dient zum Lesen des privaten Attributes prozess */
66  short getProzess();
67
68  /** dient zum Lesen (Ã¼ber ReferenzÃ¼bergabe) der entsprechenden privaten
69  Attribute. */
70  void getFaktoren(double& spgFaktor, double& tmpFaktor, double& przFaktor);
71
72  /** Diese Methode dient zum Schreiben des privaten Attributes spannung */
73  void setSpannung (double spannung);
74
75  /** Diese Methode dient zum Schreiben des privaten Attributes temp */
76  void setTemp( double temp);
77
78  /** Diese Methode dient zum Schreiben des privaten Attributes prozess */
79  void setProzess (short prozess);
80
81  /** Gibt alle berechneten Faktoren auf dem Bildschirm aus. */
82  void ausgabeFaktoren();
83
84  };
85
86  #endif // _Faktoren_

```

A.5 Faktoren.cpp

```

1 // Faktoren.cpp
2 //
3 #include "Faktoren.h"
4 #include <iostream>
5
6 using namespace std;
7
8 /** Konstruktor der Klasse. Er soll beim Anlegen der Klasse alle Attribute mit dem Wert 0 initialisieren.*/
9 Faktoren::Faktoren(){
10     spannung = 0;
11     temp = 0;
12     prozess = 0;
13     spannungFaktor = 0;
14     tempFaktor = 0;
15     prozessFaktor = 0;
16 }
17
18 /** Destruktor der Klasse.*/
19 Faktoren::~Faktoren(){}
20
21 /** Gibt alle berechneten Faktoren auf dem Bildschirm aus. */
22 void Faktoren::ausgabeFaktoren(){
23     cout << endl << "Spannungsfaktor: \t" << spannungFaktor << endl
24         << "Temperaturfaktor: \t" << tempFaktor << endl <<
25         "Prozessfaktor: \t\t" << prozessFaktor << endl;
26 }
27
28 /** dient zum Lesen (Über Referenzübergabe) der entsprechenden privaten Attribute.
29     */ //kuriose (unfertige?) Funktion
30 void Faktoren::getFaktoren(double& spgFaktor, double& tmpFaktor, double& przFaktor){
31     //
32     //erst spaeter??
33     //keine Ahnung, ob das so gemeint ist
34
35     spgFaktor = spannungFaktor;
36     tmpFaktor = tempFaktor;
37     przFaktor = prozessFaktor;
38 }
39
40 /** Beinhaltet die Werte Spannungstabelle in Form eines 2-dimensionalen Arrays.
41     Überprüft anhand des Arrays, ob der Wert vom Attribut spannung innerhalb
42     der vorgegebenen Grenzen liegt. Wenn dies nicht der Fall ist, wird eine
43     Fehlermeldung ausgegeben und false zurück gegeben. Ansonsten wird die private
44     Methode
45     berechneFaktor() mit dem Wert, dem Array und der Größe des Arrays als
46     Übergabeparameter aufgerufen. Der Rückgabewert der Methode berechneFaktor()
47     wird in dem
48     Attribut spannungFaktor gespeichert.*/
49 bool Faktoren::berechneSpannungFaktor (double spannung){
50     if ( (spannung >= 1.08) && (spannung <= 1.32)) {
51         double spgTabelle[][2] = { {1.08 , 1.121557},
52                                     {1.12 , 1.075332},

```

```

51         {1.16 , 1.035161},
52         {1.20 , 1.000000},
53         {1.24 , 0.968480},
54         {1.28 , 0.940065},
55         {1.32 , 0.9144822}};

57     spannungFaktor = Faktoren::berechneFaktor(spannung, spgTabelle, 7 );

59     return true;
60 } else {
61     cout << endl << "Fehler: Spannung ueber- oder unterschreitet die Grenzwerte!" << endl ;
62     return false;
63 }

65 }

68 /** Beinhaltet die Werte Temperaturtabelle in Form eines 2-dimensionalen Arrays.
69     Überprüft anhand des Arrays, ob der Wert vom Attribut temp innerhalb
70     der vorgegebenen Grenzen liegt. Wenn dies nicht der Fall ist, wird eine
71     Fehlermeldung ausgegeben und false zurück gegeben. Ansonsten wird die private
72     Methode
73     berechneFaktor() mit dem Wert, dem Array und der Größe des Arrays als
74     Übergabeparameter aufgerufen. Der Rückgabewert der Methode berechneFaktor()
75     wird in dem
76     Attribut tempFaktor gespeichert. */
77 bool Faktoren::berechneTempFaktor (double temp){
78     if ( (temp >= -25) && (temp <= 125)) {
79         double tempTabelle[][2] = { {-25 , 0.897498},
80             {-15 , 0.917532},
81             { 0 , 0.948338},
82             { 15 , 0.979213},
83             { 25 , 1.000000},
84             { 35 , 1.020305},
85             { 45 , 1.040540},
86             { 55 , 1.061831},
87             { 65 , 1.082983},
88             { 75 , 1.103817},
89             { 85 , 1.124124},
90             { 95 , 1.144245},
91             { 105 , 1.164563},
92             { 115 , 1.184370},
93             { 125 , 1.204966}};

94         tempFaktor = Faktoren::berechneFaktor(temp, tempTabelle, 15 );

95         return true;
96     } else {
97         cout << endl << "Fehler: Temperatur ueber- oder unterschreitet die Grenzwerte!" << endl ;
98         return false;
99     }
100 }

```

```

102  /** Beinhaltet die Werte Prozessstabelle in Form eines 2-dimensionalen Arrays. ←
103  Überprüft anhand des Arrays, ob der Wert vom Attribut prozess innerhalb
104  der vorgegebenen Grenzen liegt. Wenn dies nicht der Fall ist, wird eine ←
105  Fehlermeldung ausgegeben und false zurück gegeben. Ansonsten wird die private ←
106  Methode
107  berechneFaktor() mit dem Wert, dem Array und der Größe des Arrays als ←
108  Übergebeparameter aufgerufen. Der Rückgabewert der Methode berechneFaktor() ←
109  wird in dem
110  Attribut prozessFaktor gespeichert.*/
111  bool Faktoren::berechneProzessFaktor (short prozess){
112      if ((prozess >= 1) && (prozess <= 3)) {
113          double przTabelle[2][2] = { { 1 , 1.174235}, /** 1 = slow */
114              { 2 , 1.000000}, /** 2 = typical */
115              { 3 , 0.876148}}; /** 3 = fast */
116
117          prozessFaktor = Faktoren::berechneFaktor(prozess, przTabelle, 3 );
118
119          return true;
120      } else {
121          cout << endl << "Fehler: Prozess " << prozess << " existiert nicht !! [Slow->1, ←
122              Typical->2, Fast->3]" << endl ;
123          return false;
124      }
125  }
126
127  /** Die Methode durchsucht das übergebene Array nach dem übergebenen Wert. Wenn ←
128  der Wert im Array vorhanden ist (1. Spalte der Tabelle) wird der zugehörige
129  Faktor (2. Spalte der Tabelle) direkt zurück gegeben, ansonsten wird mit den am ←
130  nächsten liegenden Punkten eine Interpolation über die entsprechende Methode
131  gestartet und der interpolierte Wert zurück gegeben. */
132  double Faktoren::berechneFaktor (double wert, double arr[2][2], int laenge){
133
134      double Faktor;
135      double vgl = 0;
136      double untereSchranke[2] = { (arr[0][0]) , (arr[0][1]) };
137      double obereSchranke[2] = { (arr[laenge-1][0]) , (arr[laenge-1][1]) };
138
139      for (int i=0; i < laenge; i++){
140          if (wert > arr[i][0]){
141              untereSchranke[0] = arr[i][0];
142              untereSchranke[1] = arr[i][1];
143              obereSchranke[0] = arr[i+1][0];
144              obereSchranke[1] = arr[i+1][1];
145          }
146          else if (wert == arr[i][0]){
147              vgl = arr[i][0];
148              Faktor = arr[i][1];
149          }
150      }
151
152      if (wert != vgl){
153          Faktor = Faktoren::interpolation ( wert, untereSchranke[0], obereSchranke[0], ←
154              untereSchranke[1], obereSchranke[1]);
155      }
156      /** cout << untereSchranke[0] << endl << obereSchranke[0] << endl << untereSchranke[1] << ←
157      endl << obereSchranke[1] << endl; //zum testen */
158      return Faktor;

```

```
151 }

154 /** Diese Methode interpoliert einen Wert zwischen zwei vorgegebenen Punkten im 2D-
155     Raum. Dabei bestimmen x1,
156     y1 und x2, y2 jeweils die Koordinaten der zwei Punkte zwischen denen interpoliert
157     werden soll. Der
158     Übergabeparameter wert bestimmt den x-Wert des gesuchten Wertes, dabei gilt x1 <
159     wert < x2.*/
160 double Faktoren::interpolation ( double wert, double x1, double x2, double y1, double
161     y2){
162     //
163     double interpolwert = wert * (y2-y1)/(x2-x1) + (y1 - (y2-y1)/(x2-x1)*x1);
164     return interpolwert;
165 }

166 /** Diese Methode dient zum Lesen des privaten Attributes spannung */
167 double Faktoren::getSpannung(){
168     return spannung;
169 }

170 /** Diese Methode dient zum Lesen des privaten Attributes temp (die
171     eratur) */
172 double Faktoren::getTemp(){
173     return temp;
174 }

175 /** Diese Methode dient zum Lesen des privaten Attributes prozess */
176 short Faktoren::getProzess(){
177     return prozess;
178 }

179 /** Diese Methode dient zum Schreiben des privaten Attributes spannung */
180 void Faktoren::setSpannung (double spannung){
181     this->spannung = spannung;
182     berechneSpannungFaktor(spannung);
183 }

184 /** Diese Methode dient zum Schreiben des privaten Attributes temp */
185 void Faktoren::setTemp( double temp){
186     this->temp = temp;
187     berechneTempFaktor(temp);
188 }

189 /** Diese Methode dient zum Schreiben des privaten Attributes prozess */
190 void Faktoren::setProzess (short prozess){
191     this->prozess = prozess;
192     berechneProzessFaktor(prozess);
193 }
194 }
```

A.6 Bibliothek.h

```
1  #ifndef BIBLIOTHEK_H
2  #define BIBLIOTHEK_H

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"

13 #include "GatterTyp.h"
14 #include "Flipflop.h"

17 using namespace std;

19 class Bibliothek{
20 public:
21     Bibliothek();
22     virtual ~Bibliothek();

24     string getPfad(void);
25     GatterTyp* getBibElement(string typ);
26     void dateiAusgabe(void);
27     void dateiAuswerten(void);
28     bool pfadEinlesen(string pfad);

30 protected:
31 private:
32     string datei;
33     vector<GatterTyp*> bibElemente;
34     vector<Flipflop*> bibHilfe;

36     void openError(void);
37     void readError(void);
38 };

40 #endif // BIBLIOTHEK_H
```

A.7 Bibliothek.cpp

```

2  #include "Bibliothek.h"

9  Bibliothek::Bibliothek()
10 {
11     vector<GatterTyp*> bibElemente;
12 }

14 Bibliothek::~Bibliothek()
15 {
16     //dtor
17 }

19 //Dient zum Lesen des Pfads und Dateinamen, welche im Attribut datei gespeichert ←
    sind
20 string Bibliothek::getPfad(void)
21 {
22     return datei;
23 }

25 /**Dieser Methode wird ein string, des Gattertyps (z.B. inv1a), Ã¼bergegeben.
26 Sie gibt einen Zeiger auf das entsprechende Element vom Typ GatterTyp zurÃ¼ck.
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 }

44 }

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 abstÃ¼rzen, 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;

```

```

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 vom Typ
74 GatterTyp im Vektor bibElemente gespeichert werden. Die Reihenfolge ist
75 dabei nicht wichtig. Das Flipflop kann dabei am Namen erkannt werden, welcher als ↵
76 bekannt vorausgesetzt wird.
77 Das Flipflop wird in einem Element vom Typ Flipflop im Vektor bibElemente ↵
78 gespeichert. */
79 void Bibliothek::dateiAuswerten(void)
80 {
81     ifstream f(datei.c_str());
82
83     string buffer;
84     while (!f.eof())
85     {
86         getline(f,buffer);
87         ///"\r" entfernen
88         buffer.erase(buffer.size()-1);
89
90         ///von [[Bausteine]] bis Leerzeile einlesen
91         if (buffer.find("[Bausteine]")!=0)
92         {
93
94             while (!f.eof())
95             {
96                 getline(f,buffer);
97
98                 if (buffer=="\r")
99                 {
100                     debug_msg("Blockende gefunden");
101                     break;
102                 }
103
104                 ///"\r" entfernen
105                 buffer.erase(buffer.size()-1);
106
107                 ///if(buffer == "dff")
108                 {
109                     Flipflop* dummy = (new Flipflop());
110
111                     dummy->setName(buffer);

```



```

114         bibElemente.push_back(*dummy);
116         debug_msg( "ff angelegt: "<<buffer);
118     }
119     else
120     {
121         GatterTyp* dummy= new GatterTyp();
122         dummy->setName(buffer);
124         bibElemente.push_back(*dummy);
125         debug_msg( "gt angelegt: "<<buffer);
127     }*/

132     }
133 }

135 else if(buffer.find("[")==0)
136 {
137     //Klammern [ ] entfernen
138     string name = buffer.substr(1,buffer.size()-2);

141     //FF anlegen
142     if (name=="dff")
143     {
144         Flipflop *ff = new Flipflop();
145         ff->setName(name);
147         debug_msg(name <<"als FF anlegen");

150         while (!f.eof())
151         {
152             getline(f,buffer);

154             //Abbruch falls Absatz zu Ende
155             if (buffer=="\r")
156             {

158                 //FF zu bibElemente hinzufügen
159                 bibElemente.push_back((ff));

164                 debug_msg("Ende von: "<<name<<" gefunden");
165                 break;
166             }
167             //"\r" entfernen
168             buffer.erase(buffer.size()-1);

```

```

172     ///allgemeine Attribute
173     if (buffer.find("ei:") == 0)
174     {
175         ff->setEingaenge(atoi(buffer.substr(3).c_str()));
176         debug_msg( "ei init " << ff->getEingaenge());
177     }
178
179     else if (buffer.find("cl:") == 0)
180     {
181         ff->setLastKapazitaet(atoi(buffer.substr(3).c_str()));
182         debug_msg("cl init " << ff->getLastKapazitaet());
183     }
184
185     else if (buffer.find("kl:") == 0)
186     {
187         ff->setLastFaktor(atoi(buffer.substr(3).c_str()));
188         debug_msg("kl init " << ff->getLastFaktor());
189     }
190
191     else if (buffer.find("tpd0:") == 0)
192     {
193         ff->setGrundLaufzeit(atof(buffer.substr(5).c_str()));
194         debug_msg("tpd0 init " << ff->getGrundLaufzeit());
195     }
196
197     ///Flipflop Attribute
198     else if (buffer.find("tsetup:") == 0)
199     {
200         ff->setSetupTime(atoi(buffer.substr(7).c_str()));
201         debug_msg("ff testup init: " << ff->getSetupTime());
202     }
203
204     else if (buffer.find("ed:") == 0)
205     {
206         ff->setEingaenge(atoi(buffer.substr(3).c_str()));
207         debug_msg("ff ed init: " << ff->getEingaenge());
208     }
209
210     else if (buffer.find("thold:") == 0)
211     {
212         ff->setHoldTime(atoi(buffer.substr(6).c_str()));
213         debug_msg("ff thold init: " << ff->getHoldTime());
214     }
215
216     else if (buffer.find("cd:") == 0)
217     {
218         ff->setLastKapazitaet(atoi(buffer.substr(3).c_str()));
219         debug_msg("ff cd init: " << ff->getLastKapazitaet());
220     }
221
222     else if (buffer.find("tpdt:") == 0)
223     {
224         ff->setGrundLaufzeit(atof(buffer.substr(5).c_str()));
225         debug_msg("ff tpdt init: " << ff->getGrundLaufzeit());
226     }
227
228     else if (buffer.find("kl:") == 0)
229     {
230

```

```

231         ff->setLastFaktor(atoi(buffer.substr(3).c_str()));
232         debug_msg("ff kl init: "<<ff->getLastFaktor());
233
234     }
235     else if (buffer.find("ct:")!=0)
236     {
237         ff->setLastKapazitaetClock(atoi(buffer.substr(3).c_str()));
238         debug_msg("ff ct init: "<<ff->getLastKapazitaetClock());
239
240     }
241
242     else
243     {
244         if (buffer.find("#endif")!=0){
245             //Falls Attribut nicht gefunden
246             readError();
247             debug_msg(buffer<<" nicht gefunden");
248         }
249         else { bibElemente.push_back((ff));break;}
250     }
251
252 }
253
254 }
255
256 else{
257
258     GatterTyp *gt = new GatterTyp();
259
260     debug_msg(name <<"als GT anlegen");
261     gt->setName(name);
262
263
264     while (!f.eof())
265     {
266         getline(f,buffer);
267
268         //Abbruch falls Absatz zu Ende
269         if (buffer=="\r")
270         {
271
272             //GT zu bibElemente hinzufügen
273             bibElemente.push_back(gt);
274
275
276
277
278             debug_msg("Ende von: "<<name<<" gefunden");
279             break;
280         }
281         //"\r" entfernen
282         buffer.erase(buffer.size()-1);
283
284
285
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         else
315         {
316             if (buffer.find("#endf") != 0) {
317                 // Falls Attribut nicht gefunden
318                 readError();
319                 debug_msg(buffer << " nicht gefunden");
320             }
321             else break;
322         }
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 /** Speichert den Pfad zu Bibliotheksdatei im entsprechenden Attribut,
342 falls diese unter dem angegebenen Pfad vorhanden ist und sie geändert werden kann.
343 */
344 bool Bibliothek::pfadEinlesen(string pfad)

```

```
349 {
351     ifstream f(pfad.c_str());
353     if (f.good())
354     {
355         datei = pfad;
356         return true;
357     }
358     else
359     {
360         openError();
361         return false;
362     }
364 }
366 /**Ausgabe einer Fehlermeldung beim Ãndern einer Datei. */
367 void Bibliothek::openError(void)
368 {
369     cerr <<"OPEN ERROR"<<endl;
371 }
372 /**Ausgabe einer Fehlermeldung beim Lesen einer Datei.
373 */
374 void Bibliothek::readError(void)
375 {
376     cerr <<"READ ERROR"<<endl;
378 }
```

A.8 GatterTyp.h

```

1  #ifndef GATTERTYP_H
2  #define GATTERTYP_H

4  #include <string>

6  using namespace std;

8  class GatterTyp
9  {
10     public:
11         GatterTyp();
12         virtual ~GatterTyp();

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 virtual sein? auch ↔
                wenn dann sonst was wieder nicht geht..
20         void setName(string n);
21         void setGrundlaufzeit(double gl);
22         void setLastFaktor(short lf);
23         void setLastKapazitaet(short lk);
24         void setEingaenge(short ei);

27     protected:
28         string name;
29         double grundlaufzeit;
30         short lastFaktor;
31         short lastKapazitaet;
32         short eingaenge;

34     private:
35 };

37 #endif // GATTERTYP_H

```

A.9 GatterTyp.cpp

```
1  #include "GatterTyp.h"

4  GatterTyp::GatterTyp()
5  {
6      GatterTyp::eingaenge=0;
7      GatterTyp::grundLaufzeit=0;
8      GatterTyp::lastFaktor=0;
9      GatterTyp::lastKapazitaet=0;
10     GatterTyp::name="";
11 }

13 GatterTyp::~GatterTyp()
14 {
15     //dtor
16 }

18 void GatterTyp::setName(string n)
19 {
20     name=n;
21 }

23 string GatterTyp::getName(void)
24 {
25     return name;
26 }

28 double GatterTyp::getGrundLaufzeit(void)
29 {
30     return grundLaufzeit;
31 }

33 short GatterTyp::getLastFaktor(void)
34 {
35     return lastFaktor;
36 }

38 short GatterTyp::getLastKapazitaet(void)
39 {
40     return lastKapazitaet;
41 }

43 short GatterTyp::getEingaenge(void)
44 {
45     return eingaenge;
46 }

48 bool GatterTyp::getIsFlipflop(void)
49 {
50     return false;
51 }

54 void GatterTyp::setGrundLaufzeit(double gl)
55 {
56     grundLaufzeit =gl;
57 }
```

```
59 void GatterTyp::setLastFaktor(short lf)
60 {
61     lastFaktor= lf;
62 }
64 void GatterTyp::setLastKapazitaet(short lk)
65 {
66     lastKapazitaet=lk;
67 }
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>

4  Flipflop::Flipflop()
5  {
6      holdTime=0;
7      setupTime=0;
8      lastKapazitaetClock=0;
9  }

11 Flipflop::~Flipflop()
12 {
13     //dtor
14 }

15     bool Flipflop::getIsFlipflop(void){
16
17     return true;
18     }
19     short Flipflop::getSetupTime(void){
20         return setupTime;
21     }
22     short Flipflop::getHoldTime(void){
23         return holdTime;
24     }
25     void Flipflop::setSetupTime(short st){
26         setupTime=st;
27     }
28     void Flipflop::setHoldTime(short ht){
29         holdTime = ht;
30     }
31     void Flipflop::setLastKapazitaetClock(short lkc){
32         lastKapazitaetClock = lkc;
33     }
34     short Flipflop::getLastKapazitaetClock(){
35         return lastKapazitaetClock;
36     }
```

A.12 SignalListeErzeuger.h

```

1  #ifndef SIGNALLISTEERZEUGER_H
2  #define SIGNALLISTEERZEUGER_H

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"

13 using namespace std;

15 /** SignallisteErzeuger
16  Liest die komplette Datei ein, sortiert und erzeugt Liste.
17  Kritik von Lukas: sollte laut Name nur Liste erstellen.
18  */

20 class SignallisteErzeuger
21 {
22     public:
23         SignallisteErzeuger();           ///CTOR
24         virtual ~SignallisteErzeuger();  ///DTor
25         Signal* getSignal(int i);         ///gibt Instanz an der Stelle i im ↵
26         vector<Signal> signale;           zur 1/4ck
27         int readFile();                  ///Liest Datei ein und f1/4r ↵
28         sortierfunktion aus
29         int readSignalLine(signalTypen typ, int lengthBegin, string line); ↵
30         ///Liest nach Signaltyp vorsortierte Zeile ein
31         int readGateLine(string tmpLine);
32         long getFrequenz();
33         string getDatei();
34         void dateiAusgabe(void);
35         short getAnzahlSignale();
36         void setFrequenz(long freq);
37         void setDatei(string file);      ///Liest Datei NICHT ein
38         void setAnzahlSignale(short nSignals);
39     protected:
40     private:
41         vector<Signal> signale;          ///Vector mit Signal Instanzen
42         long frequenz;
43         string datei;                    ///Pfad zur Schaltnetz Datei
44         short anzahlSignale;

45 };

46 #endif // SIGNALLISTEERZEUGER_H

```

A.13 SignalListeErzeuger.cpp

```

1  #include "SignalListeErzeuger.h"

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 Menü aufgerufen
14 }

16 /**Destruktor
17 */
18 SignalListeErzeuger::~SignalListeErzeuger()
19 {
20     //dtor
21 }

23 /**dateiAusgabe
24 Öffnet die Datei die in der 'datei' Variable der Klasse gespeichert ist und gibt ↵
    die aus
25 */
26 void SignalListeErzeuger::dateiAusgabe(void)
27 {
28     ifstream f(datei.c_str());

30     string buffer;

32     int i=0;

34     if(f.good())
35     {
36         while (!f.eof())
37         {
38             getline(f,buffer);
39             cout << i<<": "<<buffer << endl;
40             i++;
41         }
42     }
43     else
44     {
45         cout << "ERR: Can not read file!";
46     }

48 }

49 /** Gibt Signal aus dem 'signale' Vektor an der im Parameter spezifizierten Stelle ↵
    zurück
50 */
51 Signal* SignalListeErzeuger::getSignal(int i) {
52     return &signale.at(i) ;
53 }

55 /**Liest die 'datei' aus und beginnt mit der Auswertung

```

```

56 */
57 int SignallisteErzeuger::readFile() {
58     signale.clear(); //Vektor 'signale' wird geleert
59     string line;
60     ifstream listfile(getDatei().data()); //Öffne Dateistream
61     Signal* bufferobj = new Signal;
62     signale.push_back( *bufferobj ); //Reserviere leeres Objekt für die ←
        CLOCK
63     if (listfile.is_open()) {
64         //debug_msg( "INFO: file is open" );
65         while (!listfile.eof()) {
66             getline(listfile,line); //liest Zeile für Zeile aus
67             if (((line.substr(0,2)) == "//") or (line == "\r") or (line == "")) {
68                 debug_msg( "INFO: drop, comment or empty line" );
69             }else if ((line.substr(0,12)) == "ARCHITECTURE") { ←
                //Wenn Kommentar, leere Zeile oder Schwachsinn drin steht, passiert ←
                gar nichts
70                 debug_msg( "INFO: drop, ARCHITECTURE shit" );
71             }else if ((line.substr(0,6)) == "ENTITY") {
72                 while (1) {
73                     getline(listfile,line);
74                     if ((line.substr(0,2)) == "//") {
75                         debug_msg( "INFO: drop, comment or empty line" );
76                     }else if ((line.substr(0,5)) == "INPUT") {
77                         debug_msg( "INFO: Found INPUT line!" );
78                         readSignalLine(eingang,5,line);
79                     }else if ((line.substr(0,6)) == "OUTPUT") {
80                         debug_msg( "INFO: Found OUTPUT line!" );
81                         readSignalLine(ausgang,6,line);
82                     }else if ((line.substr(0,7)) == "SIGNALS") {
83                         debug_msg( "INFO: Found SIGNALS line!" );
84                         readSignalLine(intern,7,line);
85                     }else if ((line.substr(0,5)) == "CLOCK") {
86                         debug_msg( "INFO: Found CLOCK line!" );
87                         string hr_frequency = line.substr(11,(line.length()-11)); ←
                            //Schneide Frequenz aus
88                         frequenz = atoi(hr_frequency.data()); ←
                            //Lese Frequenzzahl
89                         if (hr_frequency.substr(hr_frequency.size()-5,1)=="M") { ←
                            //Multipliziere ←
                                frequenz
                                frequenz = frequenz * 1000000;
90                             } else if (hr_frequency.substr(hr_frequency.size()-5,1)=="k") {
91                                 frequenz = frequenz * 1000;
92                             }
93                         bufferobj->setSignalTyp(clk);
94                         signale.at(0) = *bufferobj;
95                         debug_msg( "INFO: Set clk to: " << frequenz );
96                     }else if (line == "\r" or (line == "")){
97                         debug_msg( "INFO: Found empty line, leave ENTITY area!" );
98                         break;
99                     }else {
100                         debug_msg( "ERR: Error reading line" );
101                         break;
102                     }
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                 //Wenn Kurzschluss ←
114                 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             //AnzahlSignale auf die ←
126             Größse des Vektor setzen
127             debug_msg( "DEBUG: AnzahlSignale: " << getAnzahlSignale() ) ←
128             ;
129             return 0;
130         }else {
131             debug_msg( "ERR: Error reading line" );
132             break;
133         }
134     }
135     } else { //-----else
136         debug_msg( "ERR: Error reading headline" );
137         break;
138     }
139 }
140 } else {
141     cout << "ERR: Error opening file!";
142     cin.get();
143     return 1;
144 }
145 return 0;
146 }
147
148 int SignallisteErzeuger::readSignalLine(signalTypen typ, int lengthBegin, string ←
149 tmpLine) {
150     string tmpSignal;
151     stringstream tmpStream(tmpLine.substr(lengthBegin+1,(tmpLine.length()-(←
152         lengthBegin+2+linuxzusatz))))); //Erstellt Stream und schneidet Anfang und ←
153     Ende ab
154     while (getline(tmpStream,tmpSignal',')) { ←
155         //Trennt nach Komma
156         debug_msg( "INFO: Aktuelles Signal: " << tmpSignal );
157         unsigned int tmpSignalNo = atoi(tmpSignal.substr(1,3).c_str()); ←
158         //Lese Nummer von aktuellem Signal
159         debug_msg( "DEBUG: tmpSignalNo: " << tmpSignalNo );
160         Signal* nullObj = new Signal; ←
161         //Erzeuge leeres ←
162         Objekt
163         while (signale.size() <= tmpSignalNo) { ←
164             //Solange der Vektor kleiner ist als aktuelle Signalnummer
165             signale.push_back( *nullObj ); ←
166         }
167         //Vergrößere ←

```

```

153         }
154         signale.at(tmpSignalNo).setSignalTyp(typ);
155         //Schreibe Typ an Stelle der akt. Signalnummer in Vektor
156     }
157 }

158 int SignallisteErzeuger::readGateLine(string tmpLine) {
159     string gateNo, gatetype, tmpSignal;
160     gateNo = tmpLine.substr(0,4); //Schneide
161     //Gatenummer heraus
162     gatetype = tmpLine.substr(5,tmpLine.find("(")-5); //Schneide Gatetype
163     //abhängig von der Länge heraus
164     tmpLine = tmpLine.substr(tmpLine.find("(")+1,tmpLine.size()-tmpLine.find("(")-3-
165     linuxzusatz); //Schneide Signale heraus
166     string tmpOut = (tmpLine.substr(tmpLine.size()-2-linuxzusatz,3));
167     //Schneide Ausgang heraus
168     if (signale.at(atoi(tmpOut.c_str())).getQuelle().empty()) {
169         //Prüfe auf Kurzschluss
170         signale.at(atoi(tmpOut.c_str())).setQuelle(gateNo);
171         //Setze Quelle für Ausgangssignal
172     }
173     else {
174         return 1;
175     }
176     signale.at(atoi(tmpOut.c_str())).setQuellentyp(gatetype);
177     //Setze Quellentyp für Ausgangssignal
178     tmpLine = tmpLine.erase(tmpLine.size()-5,5);
179     //Schneide Ausgang ab
180     stringstream tmpStream(tmpLine);
181     //Erstelle String stream
182     while (getline(tmpStream,tmpSignal,',')) {
183         //Trenne nach Komma
184         debug_msg( "tmpSignal: " << tmpSignal );
185         debug_msg( "DEBUG: Vect: " << tmpSignal.substr(1,3) );
186         if (tmpSignal == "clk") {
187             signale.at(0).zielHinzufuegen(gateNo);
188         }
189         else {
190             signale.at(atoi((tmpSignal.substr(1,3)).c_str())).zielHinzufuegen(gateNo);
191             //Füge Ziele zu aktuellem Signal hinzu
192         }
193     }
194     return 0;
195 }

196 long SignallisteErzeuger::getFrequenz(){
197     return frequenz;
198 }
199 string SignallisteErzeuger::getDatei() {
200     return datei;
201 }
202 short SignallisteErzeuger::getAnzahlSignale(){
203     return anzahlSignale;
204 }
205 void SignallisteErzeuger::setFrequenz(long freq){
206     frequenz = freq;
207 }
208 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**/

4  #ifndef SIGNAL_HEADER
5  #define SIGNAL_HEADER

7  #include <string>
8  #include <iostream>
9  #include <vector>

11 enum signalTypen {eingang, intern, ausgang, unbekannt, clk};

13 using namespace std;

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 };

40 #endif
```

A.15 signals.cpp

```

1  /** Signal Class CPP File
2  created by Benibr */

4  #include "signals.h"

6  /**Signal() Ist der Konstruktor der Klasse. Er soll beim Anlegen der Klasse alle ↵
   Attribute mit dem Wert 0
7  bzw. NULL für 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 können 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 };

38 signalTypen Signal::setSignalTyp(signalTypen sigTyp) {
39     signalTyp = sigTyp;
40 };

42 void Signal::setQuelle(string gatterName) {
43     quelle = gatterName;
44 };

46 void Signal::setQuellentyp(string gatterTyp) {
47     quellenTyp = gatterTyp;
48 };

50 void Signal::setAnzahlZiele(int nZiele) {
51     anzahlZiele = nZiele;
52 };

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 Attribute mit dem
26         Wert 0 bzw. NULL für Zeiger initialisieren. Ausserdem
27         bekommt der Konstruktor einen Zeiger auf ein Element der Bibliotheksdatenbank und
28         speichert es in das Attribut typ.*/
29     SchaltwerkElement( GatterTyp* gTyp);
30     ~SchaltwerkElement();           /** Ist der Destruktor der Klasse.*/
31
32     /** Die folgenden Methoden dienen zum Lesen der privaten Attribute eines einzelnen
33         Objekts vom Typ
34         SchaltwerkElement.*/
35
36     string getName();               /** Lesen des privaten Attributes name eines
37         einzelnen Objekts vom Typ SchaltwerkElement. */
38     GatterTyp* getTyp();            /** Lesen des privaten Attributes typ eines
39         einzelnen Objekts vom Typ SchaltwerkElement. */
40     double getLaufzeitEinzelgatter(); /** Lesen des privaten Attributes
41         laufzeitEinzelgatter eines einzelnen Objekts vom Typ SchaltwerkElement. */
42     SchaltwerkElement* getNachfolger( int pos); /** Lesen des privaten Attributes
43         nachfolgerElemente eines einzelnen Objekts vom Typ SchaltwerkElement. */
44     int getAnzahlNachfolger();      /** Lesen des privaten Attributes
45         anzahlNachfolger eines einzelnen Objekts vom Typ SchaltwerkElement. */
46     short getAnzahlEingangssignale(); /** Lesen des privaten Attributes
47         anzahlEingangssignale eines einzelnen Objekts vom Typ SchaltwerkElement. */
48     bool getIsEingangsElement();    /** Lesen des privaten Attributes
49         isEingangsElement eines einzelnen Objekts vom Typ SchaltwerkElement. */
50     bool getIsAusgangsElement();    /** Lesen des privaten Attributes
51         isAusgangsElement eines einzelnen Objekts vom Typ SchaltwerkElement. */
52
53     /** Die folgenden Methoden dienen zum Schreiben der privaten Attribute eines
54         einzelnen Objekts vom Typ
55         SchaltwerkElement.*/

```

```

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                                     vom Typ ←
    SchaltwerkElement←
    .*/
48 void setAnzahlNachfolger( int anzahlN); /**Schreiben des privaten Attributes ←
    anzahlNachfolger eines einzelnen Objekts vom Typ SchaltwerkElement.*/
49 void setAnzahlEingangssignale( short anzahlE); /**Schreiben des privaten Attributes←
    anzahlEingangssignale eines einzelnen Objekts vom Typ SchaltwerkElement.*/
50 void setIsEingangselement(bool isEingangsEl); /**Schreiben des privaten Attributes ←
    isEingangselement eines einzelnen Objekts vom Typ SchaltwerkElement.*/
51 void setIsAusgangselement(bool isAusgangsEl); /**Schreiben des privaten Attributes ←
    isAusgangselement eines einzelnen Objekts vom Typ SchaltwerkElement.*/
52 void setLaufzeitEinzelgatter(double lzt); /**Schreiben des privaten Attributes ←
    laufzeitEinzelgatter eines einzelnen Objekts vom Typ SchaltwerkElement.*/
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
10 Wert 0 bzw Null für Zeiger initialisieren. Ausserdem
11 bekommt der Konstruktor einen Zeiger auf ein Element der Bibliotheksdatenbank und
12 speichert es in das Attribut typ.*/
13 SchaltwerkElement::SchaltwerkElement( GatterTyp* gTyp ){
14     name = "";
15     typ = gTyp;
16     laufzeitEinzelgatter = 0;
17     nachfolgerElemente[0] = NULL; //schäner lassen?
18     nachfolgerElemente[1] = NULL;
19     nachfolgerElemente[2] = NULL;
20     nachfolgerElemente[3] = NULL;
21     nachfolgerElemente[4] = NULL;
22     anzahlNachfolger = 0;
23     isEingangselement = false;
24     isAusgangselement = false;
25     anzahlEingangssignale = 0;
26 }
27
28 SchaltwerkElement::~SchaltwerkElement() { }    /** Destruktor der Klasse.*/
29
30 /** Die folgenden Methoden dienen zum Lesen der privaten Attribute eines einzelnen
31 Objekts vom Typ
32 SchaltwerkElement.*/
33
34 string SchaltwerkElement::getName(){           /** Lesen des privaten Attributes
35     name eines einzelnen Objekts vom Typ SchaltwerkElement. */
36     return name;
37 }
38
39 GatterTyp* SchaltwerkElement::getTyp(){         /** Lesen des privaten Attributes
40     typ eines einzelnen Objekts vom Typ SchaltwerkElement. */
41     return typ;
42 }
43
44 double SchaltwerkElement::getLaufzeitEinzelgatter(){ /** Lesen des privaten
45     Attributes laufzeitEinzelgatter eines einzelnen Objekts vom Typ
46     SchaltwerkElement. */
47     return laufzeitEinzelgatter;
48 }
49
50 SchaltwerkElement* SchaltwerkElement::getNachfolger( int pos){ /** Lesen des
51     privaten Attributes nachfolgerElemente eines einzelnen Objekts vom Typ
52     SchaltwerkElement. */
53     return nachfolgerElemente[pos];
54 }

```

```

48  int SchaltwerkElement::getAnzahlNachfolger(){ /** Lesen des privaten Attributes ↵
      anzahlNachfolger eines einzelnen Objekts vom Typ SchaltwerkElement. */
49      return anzahlNachfolger;
50  }

52  short SchaltwerkElement::getAnzahlEingangssignale(){ /** Lesen des privaten ↵
      Attributes anzahlEingangssignale eines einzelnen Objekts vom Typ ↵
      SchaltwerkElement. */
53      return anzahlEingangssignale;
54  }

56  bool SchaltwerkElement::getIsEingangsElement(){ /** Lesen des privaten Attributes ↵
      isEingangsElement eines einzelnen Objekts vom Typ SchaltwerkElement. */
57      return isEingangsElement;
58  }

60  bool SchaltwerkElement::getIsAusgangsElement(){ /** Lesen des privaten Attributes ↵
      isAusgangsElement eines einzelnen Objekts vom Typ SchaltwerkElement. */
61      return isAusgangsElement;
62  }

64  /** Die folgenden Methoden dienen zum Schreiben der privaten Attribute eines ↵
      einzelnen Objekts vom Typ
      SchaltwerkElement.*/
65  void SchaltwerkElement::setName( string n){
66      name = n;
67  }
68  }

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 Zeiger-Attribute ↵
20     mit NULL initialisieren.*/
21     ListenElement();
22
23     /** Destruktor der Klasse.*/
24     ~ListenElement();
25
26     /** Lesen des privaten Attributes schaltwerkElement eines einzelnen Objekts vom ↵
27     Typ ListenElement.*/
28     SchaltwerkElement* getSchaltwerkElement();
29
30     /** Lesen des privaten Attributes next eines einzelnen Objekts vom Typ ↵
31     ListenElement.*/
32     ListenElement* getNextElement();
33
34     /** Schreiben des privaten Attributes schaltwerkElement eines einzelnen Objekts ↵
35     vom Typ ListenElement.*/
36     void setSchaltwerkElement( SchaltwerkElement* SchaltwerkEl);
37
38     /** Schreiben des privaten Attributes next eines einzelnen Objekts vom Typ ↵
39     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 Zeiger-Attribute ↵
8     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 Objekts vom Typ ↵
18     ListenElement.*/
19 SchaltwerkElement* ListenElement::getSchaltwerkElement(){
20     return schaltwerkElement;
21 }
22
23 /** Lesen des privaten Attributes next eines einzelnen Objekts vom Typ ListenElement ↵
24     .*/
25 ListenElement* ListenElement::getNextElement(){
26     return next;
27 }
28
29 /** Schreiben des privaten Attributes schaltwerkElement eines einzelnen Objekts vom ↵
30     Typ ListenElement.*/
31 void ListenElement::setSchaltwerkElement( SchaltwerkElement* SchaltwerkEl){
32     schaltwerkElement = SchaltwerkEl;
33 }
34
35 /** Schreiben des privaten Attributes next eines einzelnen Objekts vom Typ ↵
36     ListenElement.*/
37 void ListenElement::setNextElement( ListenElement* nextEl){
38     next = nextEl;
39 }
```

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 variablen mit NULL bzw ↵
29                               0
30     ~GraphErzeuger();         /// unnuetzer Destruktor
31
32     void listeAnlegen( SignallisteErzeuger signallist); /** durchlaeuft die ↵
33                                                         Signalliste, weiss jeder Quelle ein Schaltwerk zu, uebernimmt Eigenschaften ↵
34                                                         der
35                                                         Signale und Gattertypen und ↵
36                                                         verknuepft die Schaltwerke ↵
37                                                         mit je einem ↵
38                                                         Listenelement und die ↵
39                                                         ListenElemente
40                                                         untereinander */
41     void graphErzeugen( SignallisteErzeuger signallist); /** durchlaeuft oben ↵
42                                                         angelegte Liste und verknuepft auf Grundlage der Signalliste die Schaltwerke
43                                                         miteinander */
44     void listenAusgabe ( ); // bisher nur zum testen /// gibt die Liste mit den ↵
45                                                         Schaltwerkinfos aus inkl der von graphErzeugen gefundenen ↵
46                                                         Adjazenzbeziehungen
47
48     void setBibliothek( Bibliothek* biblio); /// liest eine Bauteilbibliothek ein
49     Bibliothek* getBibliothek();           /// gibt die gespeicherte Bib zurueck /*(↵
50                                                         ungebraucht)*/
51
52     ListenElement* getStartElement(); //braucht man nicht
53     void setStartElement( ListenElement* start);
54
55     ListenElement* getEndElement();
56     void setEndElement( ListenElement* ende);

```

```
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;

63     short eingaenge = 0;
64     short ausgaenge = 0;

66     ListenElement* tmpElement = NULL;
67     anzahlSignale = signallist.getAnzahlSignale();

69     /// geht Signalliste durch
70     for (int i = 0; i < anzahlSignale; i++)
71     {
72         Signal tmpSignal = *signallist.getSignal( i );

74         if ((tmpSignal.getSignalTyp() == eingang) )    /// prueft ob Eingang
75         {
76             debug_msg("INFO: eingang gefunden");
77             eingaenge++;
78         }
79         else if (tmpSignal.getSignalTyp() == clk)    /// prueft ob Takt
80         {
81             debug_msg("INFO: clock gefunden");
82         }
83         else if ((tmpSignal.getSignalTyp() == intern) or (tmpSignal.getSignalTyp() == ←
            ausgang)) /// wenn intern oder ausgangs-signal hat das signal eine quelle←
            ,
84         {
85             /// die man in Schaltwerke ueberfuehren kann
86             if ( tmpSignal.getQuelle() != "" )    /// zum abfangen von unbenutzten ←
                Signalen
87             {
88                 GatterTyp* tmpGatter = bibliothek->getBibElement(tmpSignal.←
                    getQuellenTyp()); // kann man sich theoretisch auch sparen und ←
                    alle tmpGatter durch bibliothek->getBibElement(tmpSignal.←
                    getQuellenTyp()) ersetzen

90                 ListenElement* newListElement = new ListenElement();
91                 SchaltwerkElement* newSchaltwerkElement = new SchaltwerkElement( ←
                    tmpGatter );

93                 /// Schaltwerk uebernimmt Daten des Signals
94                 newSchaltwerkElement->setName(tmpSignal.getQuelle());
95                 newSchaltwerkElement->setAnzahlNachfolger(tmpSignal.getAnzahlZiele())←
                    ;
96                 newSchaltwerkElement->setLaufzeitEinzelgatter( tmpGatter->←
                    getGrundlaufzeit() );
97                 newSchaltwerkElement->setAnzahlEingangssignale( tmpGatter->←
                    getEingaenge());

99                 /// pruefen ob Ausgang
100                if ( tmpSignal.getSignalTyp() == ausgang )
101                {
102                    newSchaltwerkElement->setIsAusgangsElement(true);
103                    debug_msg("INFO: ausgang gefunden");
104                    ausgaenge++;
105                }

```

```

107      /// verknuepfen von Schaltwerkselement mit Listenelement
108      newListenelement->setSchaltwerkElement( newSchaltwerkElement );
109      gAnzahl++;
110      debug_msg("INFO: "<<gAnzahl<<" . Listenelement angelegt vom Typ "<< <-
          tmpSignal.getQuellentyp()<<" !");

113      /// baut die Liste auf
114      if ( startElement == NULL ) /// ist nur NULL, wenn noch kein Element <-
          der Liste existiert
115      {
116          endElement = newListenelement;
117          startElement = newListenelement;
118      }
119      else
120      {
121          tmpElement->setNextElement( newListenelement );
122          endElement = newListenelement;
123      }
124      tmpElement = newListenelement;
125
126  }
127  else // von leerer Quelle Abfrage, um ungenutzte Signale zu erkennen
128  {
129      cout << "Fehler! Unbenutztes Signal gefunden" << endl;
130      cin.ignore();
131      cin.get();
132  }
133
134  }
135
136  else // von Signaltypabfrage
137  {
138      cout << "Fehler! Unbekannter Signaltyp" << endl;
139      cin.ignore();
140      cin.get();
141  }
142  }
143  /// eingang finden
144  for (int z = 0; z < signallist.getAnzahlSignale(); z++) /// geht die <-
          Sigalliste durch
145  {
146      if ( signallist.getSignal(z)->getSignalTyp() == eingang) /// vergleicht <-
          mit Signaltypen, ob "eingang" der Signaltyp ist
147      {
148          debug_msg( "INFO: Dieses Eingangssignal hat "<<signallist.getSignal(z)-><-
          getAnzahlZiele()<<" Ziel(e)");
149          for ( int y = 0; y < signallist.getSignal(z)->getAnzahlZiele(); y++) <-
          /// durchlaeuft alle ziele dieses signals
150          {
151              string eingangsGatter = signallist.getSignal(z)->getZiel( y );
152              for (Listenelement* ptr = startElement; ptr != NULL; ptr = ptr-><-
          getNextElement()) /// und gleicht die ziele mit den <-
          schaltwerksnamen in dem
153              {
154                  if ( ptr->getSchaltwerkElement()->getName() == eingangsGatter ) <-
          /// jeweiligen listenelement ab
155                  {
156                      ptr->getSchaltwerkElement()->setIsEingangselement(true);

```

```

157         debug_msg( "INFO: "<< ptr->getSchaltwerkElement()->getName() <<
158                     <<" ist Eingang");
159     }
160 }
161 }
162 }

164 /// prueft, ob es unbeschaltete Eingaenge gibt
165 short tmpZaehler ;
166 for (ListenElement* ptr = startElement; ptr != NULL; ptr = ptr->getNextElement())
167     /// durchlaeuft die Listenelemente
168 {
169     tmpZaehler = 0;
170
171     debug_msg("INFO:-----");
172
173     for (int z = 0; z < signallist.getAnzahlSignale(); z++)    ///danach die
174         Signalliste
175     {
176         for (int r = 0; r < signallist.getSignal(z)->getAnzahlZiele(); r++) ///
177             und die Ziele eines jeden Signals
178         {
179             /// prueft, ob das Signalziel mit dem SchaltwerkElementsnamen
180             uebereinstimmt und erhoeht den Eingangszaehler bei Erfolg um 1
181             if ( signallist.getSignal(z)->getZiel(r) == ptr->getSchaltwerkElement()
182                 <<ptr->getName())
183             {
184                 tmpZaehler += 1;
185
186                 debug_msg("INFO: "<< tmpZaehler <<" . Eingang von "<< ptr->
187                     getSchaltwerkElement()->getName() <<" gefunden");
188             }
189         }
190     }
191
192     /// falls dff mit clk, hat die Bib einen Eingang zu wenig, wird hier
193     korrigiert
194     if (ptr->getSchaltwerkElement()->getTyp()->getName()=="dff") // weil der
195         clock eingang nicht mit eingelesen wird.. sollte man vlt noch aendern
196     {
197         tmpZaehler -= 1;
198     }
199
200     /// check, ob Schaltwerk und Bib fuer das jeweilige Element dieselbe Anzahl
201     Eingaenge verzeichnet haben
202     if (ptr->getSchaltwerkElement()->getTyp()->getEingaenge() != tmpZaehler)
203     {
204         cout << "Fehler!\nAnzahl Eingaenge laut Bibliothek: \t"<<ptr->
205             getSchaltwerkElement()->getTyp()->getEingaenge()<<endl
206         <<"Anzahl Eingaenge laut Schaltwerk: \t"<<tmpZaehler << endl;
207         cin.ignore();
208         cin.clear();
209         cin.get();
210     }
211 }

```

```

205 void GraphErzeuger::graphErzeugen(SignallisteErzeuger signallist)
206 {
207     /// durchlaeuft die Liste der durch ListenElemente verknuepften Schaltwerke
208     for (ListenElement* ptr = startElement; ptr != NULL; ptr = ptr->getNextElement())
209     {
210         ListenElement* tmpListenElement = ptr;
211         SchaltwerkElement* tmpSWE = tmpListenElement->getSchaltwerkElement();
212
213         /// prueft ob ein Schaltwerk maximal 5 Nachfolger besitzt
214         if ( tmpSWE->getAnzahlNachfolger() <= 5)
215         {
216
217             /// durchlaeuft die Signalliste auf der Suche nach gleichnamigen Quellen ←
der Signale und Schaltwerksnamen
218             for (int i = 0; i < signallist.getAnzahlSignale(); i++)
219             {
220
221                 Signal tmpSignal = *signallist.getSignal( i );
222
223                 if ( tmpSignal.getQuelle() == tmpSWE->getName())
224                 {
225
226                     /// bei Treffer wird wieder die Signalliste durchlaufen auf der ←
Suche nach den Zielen des gleichnamigen Signals
227                     for ( int j = 0; j < tmpSignal.getAnzahlZiele(); j++)
228                     {
229                         string folgeGatter = tmpSignal.getZiel( j );
230
231                         /// sucht zu den Zielen des Signals das entsprechende ←
Schaltwerk aus den ListenElementen
232                         for (ListenElement* ptr2 = startElement; ptr2 != NULL; ptr2 = ←
ptr2->getNextElement())
233                         {
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 }

```



```

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() == true)
273         {
274             cout<<"Schaltungsausgangselement"<< endl;
275         }
276         cout<<"Laufzeit Einzelgatter: \t"<< ptr->getSchaltwerkElement()->↵
            getLaufzeitEinzelgatter() <<endl;
277
278         cout << "Is Flipflop: \t\t"<< (( Flipflop*) (ptr->getSchaltwerkElement()-↵
            ->getTyp()) )->getIsFlipflop()<<endl;
279
280         if (ptr->getSchaltwerkElement()->getTyp()->getName()=="dff")
281         {
282             cout << "Setup-Time: \t\t" << (( Flipflop*) (ptr->getSchaltwerkElement()->↵
                getTyp()) )->getSetupTime() << endl
283             << "Hold-Time \t\t" << (( Flipflop*) (ptr->getSchaltwerkElement()->getTyp(↵
                )) )->getHoldTime()<<endl
284             << "Lastkapazitaet: \t"<< (( Flipflop*) (ptr->getSchaltwerkElement()->↵
                getTyp()) )->getLastKapazitaetClock()<<endl;
285         }
286
287         cout<<"Anzahl Eingangssignale: "<< ptr->getSchaltwerkElement()->↵
            getAnzahlEingangssignale() <<endl; // Ende Fakultative Ausgabe
288
289         /// Ausgabe der Anzahl der Folgegatter und dann der Gatter mit ihrem Namen
290         if (ptr->getSchaltwerkElement()->getAnzahlNachfolger()==1){
291             cout<<"--->Das Gatter hat "<< ptr->getSchaltwerkElement()->↵
                getAnzahlNachfolger()<<" Ziel" <<endl; /// Einzahl
292         }else if (ptr->getSchaltwerkElement()->getAnzahlNachfolger()==0){
293             cout<<"--->Das Gatter hat keine Ziele" <<endl; ↵
                /// Keine
294         }else{
295             cout<<"--->Das Gatter hat "<< ptr->getSchaltwerkElement()->↵
                getAnzahlNachfolger()<<" Ziele" <<endl; /// Mehrzahl

```

```

296     }
297
298     if (ptr->getSchaltwerkElement()->getAnzahlNachfolger()!=0) /// falls ↵
        Nachfolger existieren
299     {
300
301         string ausgabe = " ";
302         for ( int s = 0; s < ptr->getSchaltwerkElement()->getAnzahlNachfolger() ; ↵
            s++) /// werden alle Nachfolgernamen in einen Ausgabe string ↵
            geschrieben
303         {
304
305             ausgabe = ausgabe + ptr->getSchaltwerkElement()->getNachfolger(s)->↵
                getName() + " ";
306         }
307         cout << "Angeschlossene Gatter:\t"<<ausgabe <<endl;
308     }
309     else /*if (ptr->getSchaltwerkElement()->getAnzahlNachfolger()==0)*/ /// falls↵
        keine Nachfolger existieren
310     {
311         cout << ptr->getSchaltwerkElement()->getName() << " hat keine Folgegatter↵
            "<<endl;
312     }
313
314 }
315 }

```

A.22 LaufzeitAnalysator.h

```

1  #ifndef _LAUFZEITANALYSATOR_H
2  #define _LAUFZEITANALYSATOR_H

4  #include <map>
5  #include "ListenElement.h"
6  #include "Faktoren.h"
7  #include "GraphErzeuger.h"
8  #include <vector>

10 struct DFS_Daten
11 {
12     SchaltwerkElement* VaterElement;
13     double PfadLaufzeit;
14 };

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"

5  LaufzeitAnalysator::LaufzeitAnalysator(GraphErzeuger* g, Faktoren* f)
6  {

8      faktoren = f;
9      gE = g;

13     laufzeitUebergangspfad=0;
14     laufzeitAusgangspfad=0;
15     DFS_Zwischenspeicher.clear();
16     zyklusFound = false;

18 }

20 LaufzeitAnalysator::~LaufzeitAnalysator()
21 {
22     //dtor
23 }

25 void LaufzeitAnalysator::berechne_LaufzeitEinzelgatter() /// berechnet Laufzeit fuer↵
    jedes einzelne Gatter
26 {
27     double spgFaktor,
28     tmpFaktor,
29     przFaktor;

31     faktoren->getFaktoren(spgFaktor, tmpFaktor, przFaktor); /// holt sich aeussere↵
        Faktoren ueber Referenz

33     debug_msg( "INFO: SPG-F: "<<spgFaktor<<" TMP-F: "<<tmpFaktor<<" PRZ-F: "<<↵
        przFaktor<<endl<<endl);

35     for (ListenElement* ptr = gE->getStartElement(); ptr != NULL; ptr = ptr->↵
        getNextElement()) /// durchlaeuft die ListenElemente und weist jedem↵
        Schaltwerk im Listenelement seine
36     {

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;

42         for (int i = 0; i < (ptr->getSchaltwerkElement()->getAnzahlNachfolger()); i++↵
            ) /// summiert die Eingangs-C (in fF) der mit dem Ausgang verbundenen↵
            Gatter und
43         {

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);

```

```

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

53     debug_msg("INFO: Laufzeit Einzelgatter von "<< ptr->getSchaltwerkElement()->←
        getName() << ":\t"<<ptr->getSchaltwerkElement()->getLaufzeitEinzelgatter←
        ()<<endl);
54 }

56 }
57 bool LaufzeitAnalysator::DFS_startSuche(GraphErzeuger *gE)
58 {
59     zyklusFound = false;
60     vector < ListenElement * > start;

62     for(ListenElement *i = gE->getStartElement(); i!=NULL ; i=i->getNextElement())
63     {

65         if(i->getSchaltwerkElement()->getIsEingangsElement() or i->←
            getSchaltwerkElement()->getTyp()->getIsFlipflop())
66         {

69             start.push_back(i);
70         }
71     }

76     for(int h=0; h<start.size(); h++)
77     {

79         DFS(start[h]);
80     }

83     return !zyklusFound;

88 }

90 void LaufzeitAnalysator::DFS(ListenElement* s)
91 {

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;

100     }

```

```

102     DFS_Visit(s->getSchaltwerkElement(),s->getSchaltwerkElement());
105 }
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 }

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         }

132         SchaltwerkElement* v =k->getNachfolger(i);
133         debug_msg("nachfolger:"<<v->getName()<<endl);

135         if (v->getTyp()->getIsFlipflop())
136         {
137             debug_msg("ff gefunden: "<<v->getName()<<endl<<endl<<endl);

139             if (laufzeitUebergangspfad<DFS_Zwischenspeicher[k].PfadLaufzeit + k->
getLaufzeitEinzelgatter())
140             {
141                 laufzeitUebergangspfad=DFS_Zwischenspeicher[k].PfadLaufzeit + k->
getLaufzeitEinzelgatter();

143                 uebergangspfad = k->getName() + " ->" + k->getNachfolger(i)->getName()
();
144                 //cout << "Uebergangspfad: "<<uebergangspfad<<":"<<
laufzeitUebergangspfad<<endl;
145                 //String erstellen
146                 for( SchaltwerkElement* v = k ; v != s ; v = DFS_Zwischenspeicher[v].
VaterElement )
147                 {
148                     uebergangspfad.insert( 0 , ( DFS_Zwischenspeicher[v].VaterElement
->getName() + "-> " ));
149                 }

151             }
152         }

```

```

154     else if (DFS_Zwischenspeicher[v].PfadLaufzeit < (DFS_Zwischenspeicher[k].↵
155         PfadLaufzeit + k->getLaufzeitEinzelgatter()))
156     {
157         if( ( ( DFS_Zwischenspeicher[ v ].PfadLaufzeit != 0 ) or ( v== s ) ) and ↵
158             ( DFS_Zwischenspeicher[ v ].VaterElement != k) )
159         {
160             DFS_Zwischenspeicher[v].VaterElement =k;
161
162             if(zyklensuche(v))
163             {
164                 zyklusFound = true;
165                 cout << "Fehler Zyklensuche!"<<endl;
166
167             }
168         }
169         DFS_Zwischenspeicher[v].PfadLaufzeit = DFS_Zwischenspeicher[k].↵
170             PfadLaufzeit + k->getLaufzeitEinzelgatter();
171         DFS_Zwischenspeicher[v].VaterElement = k;
172
173         DFS_Visit(v,s);
174     }
175 }
176
177
178 if(k->getIsAusgangsElement() and (laufzeitAusgangspfad < (DFS_Zwischenspeicher[k↵
179     ].PfadLaufzeit + k->getLaufzeitEinzelgatter()))
180 {
181     laufzeitAusgangspfad = DFS_Zwischenspeicher[k].PfadLaufzeit + k->↵
182         getLaufzeitEinzelgatter();
183
184     ausgangspfad = k->getName();
185
186     for(SchaltwerkElement * j = k; j != s; j= DFS_Zwischenspeicher[j].↵
187         VaterElement)
188     {
189         ausgangspfad.insert(0,(DFS_Zwischenspeicher[j].VaterElement->getName() + ↵
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: " << ↵
200         laufzeitUebergangspfad << " ps" << endl;
201     cout << endl;
202     cout << "L ngster Pfad im Ausgangsschaltnetz:" << endl;
203     cout << ausgangspfad << endl << endl;
204     cout << "Maximale Laufzeit der Pfade im Ausgangsschaltnetz: " << ↵
205         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 );

208     if (maxF>1000){
209         if (maxF>1000000){

211             cout << "maxFrequenz: " << maxF/1000000 << " MHz" << endl;
212             }
213             else{
214                 cout << "maxFrequenz: " << maxF/1000 << " kHz" << endl;
215             }
216         }

218     cout << "-----" << endl;
219     if (maxF < freq){
220         cout << "Frequenz zu gross" << endl;
221     }
222     else{
223         cout << "Frequenz okay" << endl;

225     }

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 }
12 /*
13 void debug_msg(char text) {
14     #if !defined NDEBUG
15         cout << text << endl;
16     #endif
17     cout << "Blaaaa!!!!";
18 }*/
```