Karlsruher Institut für Technologie
Institut für Technik der Informationsverarbeitung

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

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

Inhaltsverzeichnis

Anhang A

Quelltext

A.1 main.cpp

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

A.2 Menue.h

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

A.3 Menue.cpp

```
1  #include "Menue.h"
2
3
4
5  /**
6  Menue Klasse
7  zustÃ¤ndig fÃ¼r Ein/Ausgabe und Navigation durch das Programm
8
9  Eine MenÃ¼fÃ¼hrung innerhalb der Konsole lÃ¤sst sich am einfachsten ↩
       realisieren, indem man zunÃ¤chst den
10 Inhalt der Konsole lÃ¶scht, die Bildschirmausgabe aktualisiert und ↩
       dann die AuswahlmÃ¶glichkeiten Ã¼ber
11 eine einfache case-Struktur abfragt.
12 FÃ¼r die Umsetzung unter Visual Studio kÃ¶nnen die folgenden Befehle ↩
       hilfreich sein:
13 system("pause"); pausiert das Programm bis eine beliebige Taste ↩
       gedrÃ¼ckt wird.
14 system("cls"); lÃ¶scht den Inhalt der Konsole.
15
16 ErwÃ¼nschte Ausgabe:
17 ****************************************
18 *     IT-Projektpraktikum WS2012/2013    *
19 *                                        *
20 * Laufzeitanalyse synchroner Schaltwerke *
21 ****************************************
22
23 (1) aeussere Faktoren
24 Spannung [Volt]: 1.2
25 Temperatur [Grad Celsius]: 55
26 Prozess (1=slow, 2=typical, 3=fast): 1
27
28 (2) Bibliothek
29 Pfad zur Bibliotheksdatei: c:\bib.txt
30
31 (3) Schaltwerk
32 Pfad zur Schaltwerksdatei: c:\csd.txt
33
34 (4) Analyse starten
35
36 (5) Programm beenden
37
38 Waehle einen Menuepunkt und bestaetige mit Enter:
39 */
40
41
42
43 Menue::Menue()
44 {
45     /**
46     ist der Konstruktor der Klasse. Erzeugt die Objekte meineFaktoren,↩
           meineBibliothek,
47     meinSignalListeErzeuger, meinGraphErzeuger und ↩
           meinLaufzeitAnalysator.
48     */
49     Faktoren meineFaktoren;
50     Bibliothek meineBibliothek;
51     SignalListeErzeuger meinSignalListeErzeuger;
```

```
52
53
54
55  //    GraphErzeuger meinGraphErzeuger = new GraphErzeuger;
56 //     LaufzeitAnalysator meinLaufzeitAnalysator = new LaufzeitAnalysator;
57  //    Signal* signale = new Signal;
58 }
59
60 Menue::~Menue()
61 {
62     /**
63     ist der Destruktor der Klasse.
64     */
65 }
66
67 void Menue::start()
68 {
69     /**
70     schreibt das HauptmenÃ¼ in die Konsole und startet die Hauptschleife, in der durch das HauptmenÃ¼
71     navigiert wird.
72     */
73
74     while(input != "5") {
75         menueKopf();
76
77         /// Faktoren HauptmenÃ¼punkt
78         cout << "(1) aeussere Faktoren \nSpannung [Volt]: " << meineFaktoren.getSpannung() << endl;
79         cout << "Temperatur [Grad Celsius]: " << meineFaktoren.getTemp() << endl;
80         cout << "Prozess (1=slow, 2=typical, 3=fast): " << meineFaktoren.getProzess() << endl;
81
82         cout << endl;
83
84         /// Bibliothek HauptmenÃ¼punkt
85         cout << "(2) Bibliothek" << endl;
86         cout << "Pfad zur Bibliotheksdatei:" << meineBibliothek.getPfad() << endl;
87
88         cout << endl;
89
90         /// Schaltwerk HauptmenÃ¼punkt
91         cout << "(3) Schaltwerk \nPfad zur Schaltwerksdatei: " << meinSignalListeErzeuger.getDatei() << endl;
92
93         cout << "\n(4) Analyse starten \n\n(5) Programm beenden\nWaehle einen Menuepunkt und bestaetige mit Enter: ";
94
95
96         getline(cin, input);
97         switch (atoi(input.c_str()))
98         {
99         case 1:
100             faktorenMenue();
101             break;
102         case 2:
```

```
103             bibliothekMenue();
104             break;
105         case 3:
106             schaltwerkMenue();
107             break;
108         case 4:
109             analyse();
110             break;
111         }
112     }
113 }
114
115 void Menue::faktorenMenue()
116 {
117     /**
118     Im UntermenÃ¼ der Ã¤usseren Faktoren sollen die Aussenbedingungen ↩
        geÃ¤ndert und die daraus resultie-
119     renden Faktoren ausgegeben werden kÃ¶nnen. Dazu wird von der ↩
        Klasse Faktoren eine Ausgabeme-
120     thode 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: ";
131
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!";
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 }
172
173 void Menue::bibliothekMenue()
174 {
175     /**
176      Im UntermenÃ¼ der Bibliothek soll der Pfad zur Bibliotheksdatei ↩
            geÃ¤ndert werden kÃ¶nnen und man
177     soll sich zur Kontrolle auch die Datei im MenÃ¼ anzeigen lassen ↩
           kÃ¶nnen. Auch die Klasse Bibliothek
178     stellt dazu eine Ausgabemethode bereit.
179     */
180
181     string pf;
182     while(input != "3") {
183         menueKopf();
184         cout << "Untermenue Bibliothek" <<endl;
185         cout << "(1) Pfad zur Bibliotheksdatei: " << meineBibliothek.↩
               getPfad() <<endl;
186         cout << "(2) Ausgabe der Bibliotheksdatei" << endl;
187         cout << "(3) Hauptmenue" << endl<< endl;
188         cout << "Waehle einen Menuepunkt und bestaetige mit Enter: ";
189
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 }
211
212 void Menue::schaltwerkMenue()
213 {
214     /**
215     Im UntermenÃ¼ des Schaltwerks soll der Pfad zur Schaltwerksdatei ↩
            verÃ¤nderbar sein. Zur Kon-
216     trolle soll diese ausgegeben werden kÃ¶nnen. Ausserdem soll eine ↩
            Liste der Signale und die Gra-
217     phstruktur ausgegeben werden kÃ¶nnen. 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() << endl;
225         cout << "(2) Ausgabe der Schaltnetzdatei" << endl;
226         cout << "(3) Ausgabe der Signale" << endl;
227         cout << "(4) Ausgabe der Graphstruktur" << endl;
228         cout << "(5) Hauptmenue\n\nWaehle einen Menuepunkt und ↩
                bestaetige mit Enter: ";
229
230         getline(cin, input);
231         switch (atoi(input.c_str())) {
232         case 1:
233             if (meineBibliothek.getPfad() != "") {
234                 cout << "Pfad eingeben: ";
235                 cin >>pf;
236                 ifstream f(pf.c_str());
237                 if(!f.good()) {
238                     cout << "Datei nicht gefunden!"<<endl;
239                     cin.ignore();
240                     cin.get();
241                 } else {
242
243                     meinSignalListeErzeuger.setDatei(pf);
244                     if (meinSignalListeErzeuger.readFile() == 21 ) {
245                         //cout << "Kurzschluss gefunden!"<< endl;
246                         cin.get();
247                     } else {
248
249                         debug_pause();
250                         meinGraphErzeuger.listeAnlegen(↩
                                meinSignalListeErzeuger);
251                         meinGraphErzeuger.graphErzeugen(↩
                                meinSignalListeErzeuger);
252                     }
253                     debug_pause();
254                 }
255             } else {
256                 cout << "\n Die Bibliothek muss als erstes geladen ↩
                        werden!";
257                 cin.get();
258             }
```

```
259             break;
260         case 2:
261             meinSignalListeErzeuger.dateiAusgabe();
262             cin.get();
263             break;
264         case 3:
265             cout << "Vector size: " << meinSignalListeErzeuger.↩
                    getAnzahlSignale() << endl;
266             cout << "Vectorcontent:"  << endl;
267             for (int i=0;i<meinSignalListeErzeuger.getAnzahlSignale();↩
                    i++) {
268                 cout << "↩
                        ----------------------------------------------\n";
269                 cout << "Nummer: " << i << endl;
270                 cout << "Signaltyp: " << meinSignalListeErzeuger.↩
                        getSignal(i)->getSignalTyp() << endl;
271                 cout << "Quelle: " << meinSignalListeErzeuger.↩
                        getSignal(i)->getQuelle() << endl;
272                 cout << "Quellentyp: " << meinSignalListeErzeuger.↩
                        getSignal(i)->getQuellenTyp() << endl;
273                 cout << "Anzahlziele: " << meinSignalListeErzeuger.↩
                        getSignal(i)->getAnzahlZiele() << endl;
274                 for (int i1=0;i1<meinSignalListeErzeuger.getSignal(i)↩
                        ->getAnzahlZiele();i1++) {
275                     cout << "-----" << meinSignalListeErzeuger.↩
                            getSignal(i)->getZiel(i1) <<endl;
276                 }
277             }
278             cin.get();
279             break;
280         case 4:
281             meinGraphErzeuger.listenAusgabe( );
282             cin.get();
283             break;
284         }
285     }
286     input.clear();
287 }
288
289 void Menue::analyse()
290 {
291     /**
292     ruft die zur Analyse benÃ¶tigten Methoden auf und gibt das ↩
            Ergebnis auf dem Bildschirm aus.
293     */
294     if ((meineFaktoren.getTemp() != 0) && (meineFaktoren.getSpannung()↩
            != 0) && (meineFaktoren.getProzess() != 0) && (↩
            meineBibliothek.getPfad() != "") && (meinSignalListeErzeuger.↩
            getDatei() != "") ) {
295         LaufzeitAnalysator lza( &meinGraphErzeuger, &meineFaktoren);
296         lza.berechne_LaufzeitEinzelgatter();
297
298         if(lza.DFS_startSuche(&meinGraphErzeuger)){
299
300             lza.maxFrequenz(meinSignalListeErzeuger.getFrequenz());
301         }
302     } else {
303         cout << "Es sind noch nicht alle benÃ¶tigten Parameter ↩
                ausgefÃ¼llt!";
```

```
304     }
305     cin.get();
306     input.clear();
307 }
308
309 void Menue::menueKopf()
310 {
311     /**
312     Gibt den Kopf der MenÃ¼s aus. Dieser bleibt in HauptmenÃ¼ und ↩
            allen UntermenÃ¼s gleich.
313     */
314     clear_screen();
315     cout << " ****************************************** \n *      IT-↩
            Projektpraktikum WS2012/2013     * \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. Ã¼berprÃ¼ft anhand des Arrays, ob der Wert↩
         vom Attribut spannung innerhalb
21   der vorgegebenen Grenzen liegt. Wenn dies nicht der Fall ist, wird ↩
        eine Fehlermeldung ausgegeben und false zurÃ¼ck gegeben. ↩
        Ansonsten wird die private Methode
22   berechneFaktor() mit dem Wert, dem Array und der GrÃ¶sse des Arrays ↩
        als Ã¼bergabeparameter aufgerufen. Der RÃ¼ckgabewert der Methode↩
         berechneFaktor() wird in dem
23   Attribut spannungFaktor gespeichert.*/
24   bool berechneSpannungFaktor (double spannung);
25
26   /** Beinhaltet die Werte Temperaturtabelle in Form eines 2-↩
        dimensionalen Arrays. Ã¼berprÃ¼ft anhand des Arrays, ob der Wert↩
         vom Attribut temp innerhalb
27   der vorgegebenen Grenzen liegt. Wenn dies nicht der Fall ist, wird ↩
        eine Fehlermeldung ausgegeben und false zurÃ¼ck gegeben. ↩
        Ansonsten wird die private Methode
28   berechneFaktor() mit dem Wert, dem Array und der GrÃ¶sse des Arrays ↩
        als Ã¼bergabeparameter aufgerufen. Der RÃ¼ckgabewert der Methode↩
         berechneFaktor() wird in dem
29   Attribut tempFaktor gespeichert. */
30   bool berechneTempFaktor (double temp);
31
32   /** Beinhaltet die Werte Prozesstablle in Form eines 2-dimensionalen↩
         Arrays. Ã¼berprÃ¼ft anhand des Arrays, ob der Wert vom Attribut↩
         prozess innerhalb
33   der vorgegebenen Grenzen liegt. Wenn dies nicht der Fall ist, wird ↩
        eine Fehlermeldung ausgegeben und false zurÃ¼ck gegeben. ↩
        Ansonsten wird die private Methode
34   berechneFaktor() mit dem Wert, dem Array und der GrÃ¶sse des Arrays ↩
        als Ã¼bergabeparameter aufgerufen. Der RÃ¼ckgabewert der Methode↩
         berechneFaktor() wird in dem
35   Attribut prozessFaktor gespeichert.*/
36   bool berechneProzessFaktor (short prozess);
37
38   /** Die Methode durchsucht das Ã¼bergebene Array nach dem ↩
        Ã¼bergebenen Wert. Wenn der Wert im Array vorhanden ist (1. ↩
```

```
      Spalte der Tabelle) wird der zugehÃ¶rige
39    Faktor (2. Spalte der Tabelle) direkt zurÃ¼ckgegeben, ansonsten wird
        mit den am nÃ¤chsten liegenden Punkten eine Interpolation Ã¼ber
        die entsprechende Methode
40    gestartet und der interpolierte Wert zurÃ¼ckgegeben. */
41    double berechneFaktor (double wert, double arr[][2], int laenge);
42
43    /** Diese Methode interpoliert einen Wert zwischen zwei vorgegebenen
        Punkten im 2D-Raum. Dabei bestimmen x1,
44    y1 und x2, y2 jeweils die Koordinaten der zwei Punkte zwischen denen
        interpoliert werden soll. Der
45    Ã¼bergabeparameter wert bestimmt den x-Wert des gesuchten Wertes,
        dabei gilt x1 < wert < x2.*/
46    double interpolation ( double wert, double x1, double x2, double y1,
        double y2);
47
48  public:
49
50    /** Konstruktor der Klasse. Er soll beim Anlegen der Klasse alle
        Attribute mit dem Wert 0 initialisieren.*/
51    Faktoren();
52
53    /** Destruktor der Klasse. */
54    ~Faktoren();
55
56    /** Diese Methode dient zum Lesen des privaten Attributes spannung
        */
57    double getSpannung();
58
59    /** Diese Methode dient zum Lesen des privaten Attributes temp (die
        Temperatur) */
60    double getTemp();
61
62    /** Diese Methode dient zum Lesen des privaten Attributes prozess */
63    short getProzess();
64
65    /** dient zum Lesen (Ã¼ber ReferenzÃ¼bergabe) der entsprechenden
        privaten Attribute. */
66    void getFaktoren(double& spgFaktor, double& tmpFaktor, double&
        przFaktor);
67
68    /** Diese Methode dient zum Schreiben des privaten Attributes
        spannung */
69    void setSpannung (double spannung);
70
71    /** Diese Methode dient zum Schreiben des privaten Attributes temp
        */
72    void setTemp( double temp);
73
74    /** Diese Methode dient zum Schreiben des privaten Attributes
        prozess */
75    void setProzess (short prozess);
76
77    /** Gibt alle berechneten Faktoren auf dem Bildschirm aus. */
78    void ausgabeFaktoren();
79
80  };
81
82
```

```
83 #endif // _Faktoren_
```

A.5 Faktoren.cpp

```
1  // Faktoren.cpp
2  //
3  #include "Faktoren.h"
4  #include <iostream>
5
6  using namespace std;
7
8
9  /** Konstruktor der Klasse. Er soll beim Anlegen der Klasse alle ↩
       Attribute mit dem Wert 0 initialisieren.*/
10 Faktoren::Faktoren(){
11   spannung = 0;
12   temp = 0;
13   prozess = 0;
14   spannungFaktor = 0;
15   tempFaktor = 0;
16   prozessFaktor = 0;
17 }
18
19
20 /** Destruktor der Klasse.*/
21 Faktoren::~Faktoren(){}
22
23 /** Gibt alle berechneten Faktoren auf dem Bildschirm aus. */
24 void Faktoren::ausgabeFaktoren(){
25   cout << endl << "Spannungsfaktor: \t" << spannungFaktor <<
26       endl << "Temperaturfaktor: \t" << tempFaktor << endl <<
27           "Prozessfaktor: \t\t" << prozessFaktor << endl;
28 }
29
30
31 /** dient zum Lesen (Ã¼ber ReferenzÃ¼bergabe) der entsprechenden ↩
       privaten Attribute. */           //kuriose (unfertige?) Funktion
32 void Faktoren::getFaktoren(double& spgFaktor, double& tmpFaktor, ↩
       double& przFaktor){
33   //
34   //erst spaeter??
35   //keine Ahnung, ob das so gemeint ist
36
37   spgFaktor = spannungFaktor;
38   tmpFaktor = tempFaktor;
39   przFaktor = prozessFaktor;
40 }
41
42
43 /** Beinhaltet die Werte Spannungstabelle in Form eines 2-↩
       dimensionalen Arrays. Ã¼berprÃ¼ft anhand des Arrays, ob der Wert ↩
       vom Attribut spannung innerhalb
44 der vorgegebenen Grenzen liegt. Wenn dies nicht der Fall ist, wird ↩
       eine Fehlermeldung ausgegeben und false zurÃ¼ck gegeben. Ansonsten↩
        wird die private Methode
45 berechneFaktor() mit dem Wert, dem Array und der GrÃ¶sse des Arrays ↩
       als Ã¼bergabeparameter aufgerufen. Der RÃ¼ckgabewert der Methode ↩
       berechneFaktor() wird in dem
46 Attribut spannungFaktor gespeichert.*/
47 bool Faktoren::berechneSpannungFaktor (double spannung){
48   if  ( (spannung >= 1.08) && (spannung <= 1.32)) {
```

```
49      double spgTabelle[][2] = {  {1.08 , 1.121557},
50                      {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}};
56
57      spannungFaktor = Faktoren::berechneFaktor(spannung, spgTabelle, 7 ↩
            );
58
59      return true;
60    } else {
61      cout << endl << "Fehler: Spannung ueber- oder unterschreitet die ↩
            Grenzwerte!" << endl ;
62      return false;
63    }
64
65  }
66
67
68  /** Beinhaltet die Werte Temperaturtabelle in Form eines 2-↩
        dimensionalen Arrays. Ã¼berprÃ¼ft anhand des Arrays, ob der Wert ↩
        vom Attribut temp innerhalb
69  der vorgegebenen Grenzen liegt. Wenn dies nicht der Fall ist, wird ↩
        eine Fehlermeldung ausgegeben und false zurÃ¼ck gegeben. Ansonsten↩
         wird die private Methode
70  berechneFaktor() mit dem Wert, dem Array und der GrÃ¶sse des Arrays ↩
        als Ã¼bergabeparameter aufgerufen. Der RÃ¼ckgabewert der Methode ↩
        berechneFaktor() wird in dem
71  Attribut tempFaktor gespeichert. */
72  bool Faktoren::berechneTempFaktor (double temp){
73    if  ( (temp >= -25) && (temp <= 125)) {
74      double tempTabelle[][2] = { {-25 , 0.897498},
75                      {-15 , 0.917532},
76                      {  0 , 0.948338},
77                      { 15 , 0.979213},
78                      { 25 , 1.000000},
79                      { 35 , 1.020305},
80                      { 45 , 1.040540},
81                      { 55 , 1.061831},
82                      { 65 , 1.082983},
83                      { 75 , 1.103817},
84                      { 85 , 1.124124},
85                      { 95 , 1.144245},
86                      { 105 , 1.164563},
87                      { 115 , 1.184370},
88                      { 125 , 1.204966}};
89
90      tempFaktor = Faktoren::berechneFaktor(temp, tempTabelle, 15 );
91
92      return true;
93    } else {
94      cout << endl << "Fehler: Temperatur ueber- oder unterschreitet die↩
            Grenzwerte!" << endl ;
95      return false;
96    }
97
98  }
```

```
99
100
101
102 /** Beinhaltet die Werte Prozesstablle in Form eines 2-dimensionalen Arrays. Ã¼berprÃ¼ft anhand des Arrays, ob der Wert vom Attribut prozess innerhalb
103 der vorgegebenen Grenzen liegt. Wenn dies nicht der Fall ist, wird eine Fehlermeldung ausgegeben und false zurÃ¼ck gegeben. Ansonsten wird die private Methode
104 berechneFaktor() mit dem Wert, dem Array und der GrÃ¶sse des Arrays als Ã¼bergabeparameter aufgerufen. Der RÃ¼ckgabewert der Methode berechneFaktor() wird in dem
105 Attribut prozessFaktor gespeichert.*/
106 bool Faktoren::berechneProzessFaktor (short prozess){
107   if  ((prozess >= 1) && (prozess <= 3)) {
108     double przTabelle[][2] = {  { 1 , 1.174235},  /**  1 = slow    */
109                   { 2 , 1.000000},  /**  2 = typical */
110                   { 3 , 0.876148}}; /**  3 = fast    */
111
112     prozessFaktor = Faktoren::berechneFaktor(prozess, przTabelle, 3 );
113
114     return true;
115   } else {
116     cout << endl << "Fehler: Prozess "<< prozess <<" existiert nicht !! [Slow->1, Typical->2, Fast->3]" << endl ;
117     return false;
118   }
119
120 }
121
122
123 /** Die Methode durchsucht das Ã¼bergebene Array nach dem Ã¼bergebenen Wert. Wenn der Wert im Array vorhanden ist (1. Spalte der Tabelle) wird der zugehÃ¶rige
124 Faktor (2. Spalte der Tabelle) direkt zurÃ¼ckgegeben, ansonsten wird mit den am nÃ¤chsten liegenden Punkten eine Interpolation Ã¼ber die entsprechende Methode
125 gestartet und der interpolierte Wert zurÃ¼ckgegeben. */
126 double Faktoren::berechneFaktor (double wert, double arr[][2], int laenge){
127
128   double Faktor;
129   double vgl = 0;
130   double untereSchranke[2] = { (arr[0][0]) , (arr[0][1]) };
131   double obereSchranke[2] = { (arr[laenge-1][0]) , (arr[laenge-1][1]) };
132
133   for (int i=0; i < laenge; i++){
134     if (wert > arr[i][0]){
135       untereSchranke[0] = arr[i][0];
136       untereSchranke[1] = arr[i][1];
137       obereSchranke[0] = arr[i+1][0];
138       obereSchranke[1] = arr[i+1][1];
139     }
140     else if (wert == arr[i][0]){
141       vgl = arr[i][0];
142       Faktor = arr[i][1];
143     }
144   }
```

```
145
146   if (wert != vgl){
147     Faktor = Faktoren::interpolation ( wert, untereSchranke[0], ↩
          obereSchranke[0], untereSchranke[1], obereSchranke[1]);
148   }
149   /** cout <<untereSchranke[0]<<endl<< obereSchranke[0]<< endl<<↩
        untereSchranke[1]<<endl<< obereSchranke[1] <<endl; //zum testen ↩
        */
150   return Faktor;
151 }
152
153
154 /** Diese Methode interpoliert einen Wert zwischen zwei vorgegebenen ↩
      Punkten im 2D-Raum. Dabei bestimmen x1,
155 y1 und x2, y2 jeweils die Koordinaten der zwei Punkte zwischen denen ↩
      interpoliert werden soll. Der
156 Ã¼bergabeparameter wert bestimmt den x-Wert des gesuchten Wertes, ↩
      dabei gilt x1 < wert < x2.*/
157 double Faktoren::interpolation ( double wert, double x1, double x2, ↩
      double y1, double y2){
158   //
159   double interpolwert = wert * (y2-y1)/(x2-x1) + (y1 - (y2-y1)/(x2-x1)↩
        *x1);
160   return interpolwert;
161 }
162
163
164 /** Diese Methode dient zum Lesen des privaten Attributes spannung */
165 double Faktoren::getSpannung(){
166   return spannung;
167 }
168
169
170 /** Diese Methode dient zum Lesen des privaten Attributes temp (die
171 eratur) */
172 double Faktoren::getTemp(){
173   return temp;
174 }
175
176
177 /** Diese Methode dient zum Lesen des privaten Attributes prozess */
178 short Faktoren::getProzess(){
179   return prozess;
180 }
181
182
183 /** Diese Methode dient zum Schreiben des privaten Attributes spannung↩
        */
184 void Faktoren::setSpannung (double spannung){
185    this->spannung = spannung;
186    berechneSpannungFaktor(spannung);
187 }
188
189
190  /** Diese Methode dient zum Schreiben des privaten Attributes temp */
191 void Faktoren::setTemp( double temp){
192   this->temp = temp;
193   berechneTempFaktor(temp);
194 }
```

```
195
196
197 /** Diese Methode dient zum Schreiben des privaten Attributes prozess ↩
        */
198 void Faktoren::setProzess (short prozess){
199   this->prozess = prozess;
200   berechneProzessFaktor(prozess);
201 }
```

A.6 Bibliothek.h

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

A.7 Bibliothek.cpp

```
1
2  #include "Bibliothek.h"
3
4
5
6
7
8
9  Bibliothek::Bibliothek()
10 {
11     vector<GatterTyp*> bibElemente;
12 }
13
14 Bibliothek::~Bibliothek()
15 {
16     //dtor
17 }
18
19 //Dient zum Lesen des Pfads und Dateinamen, welche im Attribut datei ↩
       gespeichert sind
20 string Bibliothek::getPfad(void)
21 {
22     return datei;
23 }
24
25 /**Dieser Methode wird ein string, des Gattertyps (z.B. inv1a), ↩
       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     }
43
44 }
45
46 /**Ausgabe der Datei auf dem Bildschirm, dabei sollen die Zeilen ↩
       durchnummeriert werden.
47 Dabei soll, falls die Datei nicht vorhanden ist oder ein Fehler beim ↩
       Lesen auftritt,
48 das Programm nicht 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;
56
57      if(f.good())
58      {
59          while (!f.eof())
60          {
61              getline(f,buffer);
62              cout << i<<": "<<buffer<<endl;
63              i++;
64          }
65      }
66      else
67      {
68          openError();
69      }
70
71  }
72  /**Die Methode dient zum Einlesen und Auswerten der Bibliotheksdatei.
73  Dabei soll jeder in der Datei beschriebene Gattertyp in einem Element ↩
        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 bekannt vorausgesetzt wird.
76  Das Flipflop wird in einem Element vom Typ Flipflop im Vektor ↩
        bibElemente gespeichert. */
77  void Bibliothek::dateiAuswerten(void)
78  {
79      ifstream f(datei.c_str());
80
81      string buffer;
82      while (!f.eof())
83      {
84          getline(f,buffer);
85          //"\r" entfernen
86          buffer.erase(buffer.size()-1);
87
88          //von [[Bausteine]] bis Leerzeile einlesen
89          if(buffer.find("[[Bausteine]]")==0)
90          {
91
92              while (!f.eof())
93              {
94                  getline(f,buffer);
95
96                  if(buffer=="\r")
97                  {
98
99                      debug_msg("Blockende gefunden");
100                     break;
101                 }
102
103                 //"\r" entfernen
104                 buffer.erase(buffer.size()-1);
105
106
107
```

```
108                 /*if(buffer =="dff")
109                 {
110                     Flipflop* dummy = (new Flipflop());
111 
112                     dummy->setName(buffer);
113 
114                     bibElemente.push_back(*dummy);
115 
116                     debug_msg( "ff angelegt: "<<buffer);
117 
118                 }
119                 else
120                 {
121                     GatterTyp* dummy= new GatterTyp();
122                     dummy->setName(buffer);
123 
124                     bibElemente.push_back(*dummy);
125                     debug_msg( "gt angelegt: "<<buffer);
126 
127                 }*/
128 
129 
130 
131 
132         }
133     }
134 
135     else if(buffer.find("[")==0)
136     {
137         //Kalmmern [ ] entfernen
138         string name = buffer.substr(1,buffer.size()-2);
139 
140 
141             //FF anlegen
142             if(name=="dff")
143             {
144                 Flipflop *ff = new Flipflop();
145                 ff->setName(name);
146 
147                 debug_msg(name <<"als FF anlegen");
148 
149 
150                 while (!f.eof())
151                 {
152                     getline(f,buffer);
153 
154                     //Abbruch falls Absatz zu Ende
155                     if(buffer=="\r")
156                     {
157 
158                         //FF zu bibElemente hinzfÃ¼gen
159                          bibElemente.push_back((ff));
160 
161 
162 
163 
164                         debug_msg("Ende von: "<<name<<" gefunden")↩
                            ;
165                         break;
```

```
166                         }
167                         //"\r" entfernen
168                         buffer.erase(buffer.size()-1);
169
170
171
172                         ///*allgemeine Attribute
173                         if(buffer.find("ei:")==0)
174                         {
175                             ff->setEingaenge(atoi(buffer.substr(3).↩
                                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
198                         ///*Flipflop Attribute
199                         else if(buffer.find("tsetup:")==0)
200                         {
201                             ff->setSetupTime(atoi(buffer.substr(7).↩
                                c_str()));
202                             debug_msg("ff testup init: "<<ff->↩
                                getSetupTime());
203
204                         }
205                         else if(buffer.find("ed:")==0)
206                         {
207                             ff->setEingaenge(atoi(buffer.substr(3).↩
                                c_str()));
208                             debug_msg("ff ed init: "<<ff->getEingaenge↩
                                ());
209
210                         }
211                         else if(buffer.find("thold:")==0)
212                         {
```

```
213                             ff->setHoldTime(atoi(buffer.substr(6).↩
                                    c_str()));
214                             debug_msg("ff thold init: "<<ff->↩
                                    getHoldTime());
215
216                         }
217                         else if(buffer.find("cd:")==0)
218                         {
219                             ff->setLastKapazitaet(atoi(buffer.substr↩
                                    (3).c_str()));
220                             debug_msg("ff cd init: "<<ff->↩
                                    getLastKapazitaet());
221
222                         }
223                         else if(buffer.find("tpdt:")==0)
224                         {
225                             ff->setGrundLaufzeit(atof(buffer.substr(5)↩
                                    .c_str()));
226                             debug_msg("ff tpdt init: "<<ff->↩
                                    getGrundLaufzeit());
227
228                         }
229                         else if(buffer.find("kl:")==0)
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("#endf")!=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
257                 else{
258
259                     GatterTyp *gt = new GatterTyp();
260
261                     debug_msg(name <<"als GT anlegen");
```

```
262                     gt->setName(name);
263
264
265
266                                 while (!f.eof())
267                     {
268                         getline(f,buffer);
269
270                         //Abbruch falls Absatz zu Ende
271                         if(buffer=="\r")
272                         {
273
274                             //GT zu bibElemente hinzfÃ¼gen
275                             bibElemente.push_back(gt);
276
277
278
279
280
281                             debug_msg("Ende von: "<<name<<" gefunden")↩
                                ;
282                             break;
283                         }
284                         //"\r" entfernen
285                         buffer.erase(buffer.size()-1);
286
287
288
289                         ///*allgemeine Attribute
290                         if(buffer.find("ei:")==0)
291                         {
292                             gt->setEingaenge(atoi(buffer.substr(3).↩
                                c_str()));
293                             debug_msg( "ei init "<<gt->getEingaenge())↩
                                ;
294                         }
295
296                         else if(buffer.find("cl:")==0)
297                         {
298                             gt->setLastKapazitaet(atoi(buffer.substr↩
                                (3).c_str()));
299                             debug_msg("cl init "<<gt->↩
                                getLastKapazitaet());
300                         }
301
302                         else if(buffer.find("kl:")==0)
303                         {
304                             gt->setLastFaktor(atoi(buffer.substr(3).↩
                                c_str()));
305                             debug_msg("kl init "<<gt->getLastFaktor())↩
                                ;
306                         }
307
308                         else if(buffer.find("tpd0:")==0)
309                         {
310                             gt->setGrundLaufzeit(atof(buffer.substr(5)↩
                                .c_str()));
311                             debug_msg("tpd0 init "<<gt->↩
                                getGrundLaufzeit());
```

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

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

A.8 GatterTyp.h

```
1  #ifndef GATTERTYP_H
2  #define GATTERTYP_H
3
4  #include <string>
5
6  using namespace std;
7
8  class GatterTyp
9  {
10     public:
11         GatterTyp();
12         virtual ~GatterTyp();
13
14         string getName(void);
15         double getGrundLaufzeit(void);
16         short getLastFaktor(void);
17         short getLastKapazitaet(void);
18         short getEingaenge(void);
19         virtual bool getIsFlipflop(void);           // muss das nicht ↩
               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);
25
26
27     protected:
28         string name;
29         double grundLaufzeit;
30         short lastFaktor;
31         short lastKapazitaet;
32         short eingaenge;
33
34     private:
35 };
36
37 #endif // GATTERTYP_H
```

A.9 GatterTyp.cpp

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

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

A.10 Flipflop.h

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

A.11 Flipflop.cpp

```
1  #include "Flipflop.h"
2  #include <iostream>
3
4  Flipflop::Flipflop()
5  {
6      holdTime=0;
7      setupTime=0;
8      lastKapazitaetClock=0;
9  }
10
11 Flipflop::~Flipflop()
12 {
13     //dtor
14 }
15         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
3
4  #include <iostream>
5  #include <string>
6  #include <sstream>
7  #include <fstream>
8  #include <cstdlib>
9  #include <vector>
10 #include "signals.h"
11 #include "cross-compatibility.h"
12
13 using namespace std;
14
15 /** SignalListeErzeuger
16 Liest die komplette Datei ein, sortiert und erzeugt Liste.
17 Kritik von Lukas: sollte laut Name nur Liste erstellen.
18 **/
19
20 class SignalListeErzeuger
21 {
22     public:
23         SignalListeErzeuger();            ///CTor
24         virtual ~SignalListeErzeuger();              ///DTor
25         Signal* getSignal(int i);                    ///gibt Instanz an↩
               der Stelle i im vector signale zurÃ¼ck
26         int readFile();                              ///Liest Datei ein↩
               und fÃ¼r sortierfunktion aus
27         int readSignalLine(signalTypen typ, int lengthBegin, string ↩
              line);                              ///Liest nach Signaltyp ↩
              vorsortierte Zeile ein
28         int readGateLine(string tmpLine);
29         long getFrequenz();
30         string getDatei();
31         void dateiAusgabe(void);
32         short getAnzahlSignale();
33         void setFrequenz(long freq);
34         void setDatei(string file);                  ///Liest Datei ↩
              NICHT ein
35         void setAnzahlSignale(short nSignals);
36     protected:
37     private:
38         vector <Signal> signale;                     ///Vector mit ↩
              Signal Instanzen
39         long frequenz;
40         string datei;                                ///Pfad zur ↩
              Schaltnetz Datei
41         short anzahlSignale;
42 };
43
44 #endif // SIGNALLISTEERZEUGER_H
```

A.13 SignalListeErzeuger.cpp

```
1  #include "SignalListeErzeuger.h"
2
3  /**Konstruktor
4  Setzt alle Variablen auf 0
5  **/
6  SignalListeErzeuger::SignalListeErzeuger()
7  {
8      //ctor
9      anzahlSignale = 0;
10     frequenz = 0;
11     datei="";
12     /*setDatei(file);
13     readFile();*/ ///Manuell im MenÃ¼ aufrufen
14 }
15
16 /**Destruktor
17 **/
18 SignalListeErzeuger::~SignalListeErzeuger()
19 {
20     //dtor
21 }
22
23 /**dateiAusgabe
24 Ã¶ffnet 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 zurÃ¼ck
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());           ///Ã¶ffne Dateistream
61     Signal* bufferobj = new Signal;
62     signale.push_back( *bufferobj );                ///Reserviere leeres Objekt fÃ¼r die CLOCK
63     if (listfile.is_open()) {
64         //debug_msg( "INFO: file is open" );
65         while (!listfile.eof()) {
66             getline(listfile,line);                 ///liest Zeile fÃ¼r 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") {
90                             ///Multipliziere frequenz
91                             frequenz = frequenz * 1000000;
92                         } else if (hr_frequency.substr(hr_frequency.size()-5,1)=="k") {
93                             frequenz = frequenz * 1000;
94                         }
95                         bufferobj->setSignalTyp(clk);
96                         signale.at(0) = *bufferobj;
97                         debug_msg( "INFO: Set clk to:  " << frequenz );
                    }else if (line == "\r" or (line == "")){
```

```
98                          debug_msg( "INFO: Found empty line, leave ↩
                               ENTITY area!" );
99                          break;
100                     }else {
101                         debug_msg( "ERR: Error reading line" );
102                         break;
103                     }
104                 }
105             }else if ((line.substr(0,5)) == "BEGIN") {
106                 while (1) {
107                     getline(listfile,line);
108                     if ((line.substr(0,2)) == "//") {
109                         debug_msg( "comment" );
110                     }else if ((line.substr(0,1)) == "g") {
111                         debug_msg( "INFO: Found GATE line!" );
112                         if (readGateLine(line) == 1 ) { ↩
                                                                    ↩
                                ///Wenn Kurzschluss bereits vorhanden
113                              cout << "ERR: Short curcuit"  << endl;
114                              cin.get();
115                              return 21;
116                         }
117                     }else if ((line.substr(0,6)) == "END") {
118                         debug_msg( "INFO: Found END line!" );
119                             signalTypen tmpsig;
120                             tmpsig = signale.at(0).getSignalTyp();
121                             debug_msg( "DEBUG "<< tmpsig );
122                             setAnzahlSignale(signale.size()); ↩
                                                                    ↩
                                ///AnzahlSignale auf die GrÃ¶sse des ↩
                                Vektor setzen
123                             debug_msg( "DEBUG: AnzahlSignale: " << ↩
                                getAnzahlSignale() );
124                         return 0;
125                     }else {
126                         debug_msg( "ERR: Error reading line" );
127                         break;
128                     }
129                 }
130             } else {//-----------------------------------------else
131                 debug_msg( "ERR: Error reading headline" );
132                 break;
133             }
134         }
135     } else {
136         cout << "ERR: Error opening file!";
137         cin.get();
138         return 1;
139     }
140     return 0;
141 }
142
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 ); ↩
                                                                        ↩
                  ///VergrÃ¶ssere Vektor
153         }
154         signale.at(tmpSignalNo).setSignalTyp(typ); ↩
                                                    ///Schreibe Typ ↩
               an Stelle der akt. Signalnummer in Vektor
155     }
156 }
157
158 int SignalListeErzeuger::readGateLine(string tmpLine) {
159     string gateNo, gatetype, tmpSignal;
160     gateNo = tmpLine.substr(0,4); ↩
                                                ///Schneide Gatenummer ↩
           heraus
161     gatetype = tmpLine.substr(5,tmpLine.find("(")-5); ↩
                            ///Schneide Gatetyp abhÃ¤ngig von der LÃ¤nge↩
            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()) { ↩
                                              ///PrÃ¼fe auf Kurzschluss
165         signale.at(atoi(tmpOut.c_str())).setQuelle(gateNo); ↩
                                                    ///Setze Quelle ↩
               fÃ¼r Ausgangssignal
166     }
167     else {
168         return 1;
169     }
170     signale.at(atoi(tmpOut.c_str())).setQuellentyp(gatetype); ↩
                                                ///Setze Quelletyp ↩
           fÃ¼r 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);                ///FÃ¼ge Ziele ↩
                zu aktuellem Signal hinzu
181         }
182     }
183     return 0;
184 }
185
186 long SignalListeErzeuger::getFrequenz(){
187     return frequenz;
188 }
189 string SignalListeErzeuger::getDatei() {
190     return datei;
191 }
192 short SignalListeErzeuger::getAnzahlSignale(){
193     return anzahlSignale;
194 }
195 void SignalListeErzeuger::setFrequenz(long freq){
196     frequenz = freq;
197 }
198 void SignalListeErzeuger::setDatei(string file){
199     datei = file;
200 }
201 void SignalListeErzeuger::setAnzahlSignale(short nSignals){
202     anzahlSignale = nSignals;
203 }
```

A.14 signals.h

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

A.15 signals.cpp

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

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

A.16 SchaltwerkElement.h

```
1  // SchaltwerkElement.h
2  //
3  //
4
5  #ifndef _SchaltwerkElement_
6  #define _SchaltwerkElement_
7
8  #include "GatterTyp.h"
9  #include <string>
10
11
12 class SchaltwerkElement {
13 private:
14   string        name;
15   GatterTyp*      typ;
16   double        laufzeitEinzelgatter;
17   SchaltwerkElement*  nachfolgerElemente[5];
18   int          anzahlNachfolger;
19   bool         isEingangsElement;
20   bool         isAusgangsElement;
21   short        anzahlEingangssignale;
22
23 public:
24
25   /** Konstruktor der Klasse. Er soll beim Anlegen der Klasse alle ↩
       Attribute mit dem Wert 0 bzw. NULL fÃ¼r Zeiger initialisieren. ↩
       Ausserdem
26   bekommt der Konstruktor einen Zeiger auf ein Element der ↩
       Bibliotheksdatenbank und speichert es in das Attribut typ.*/
27   SchaltwerkElement( GatterTyp* gTyp);
28     ~SchaltwerkElement();                    /** Ist der Destruktor der↩
         Klasse.*/
29
30   /** Die folgenden Methoden dienen zum Lesen der privaten Attribute ↩
       eines einzelnen Objekts vom Typ
31     SchaltwerkElement.*/
32
33     string getName();                        /** Lesen des privaten ↩
         Attributes name eines einzelnen Objekts vom Typ ↩
         SchaltwerkElement. */
34   GatterTyp* getTyp();                    /** Lesen des privaten ↩
       Attributes typ eines einzelnen Objekts vom Typ SchaltwerkElement↩
       . */
35     double getLaufzeitEinzelgatter();       /** Lesen des privaten ↩
         Attributes laufzeitEinzelgatter eines einzelnen Objekts vom ↩
         Typ SchaltwerkElement. */
36   SchaltwerkElement* getNachfolger( int pos);  /** Lesen des privaten ↩
       Attributes nachfolgerElemente eines einzelnen Objekts vom Typ ↩
       SchaltwerkElement. */
37   int getAnzahlNachfolger();              /** Lesen des privaten ↩
       Attributes anzahlNachfolger eines einzelnen Objekts vom Typ ↩
       SchaltwerkElement. */
38   short getAnzahlEingangssignale();       /** Lesen des privaten ↩
       Attributes anzahlEingangssignale eines einzelnen Objekts vom Typ↩
        SchaltwerkElement. */
39   bool getIsEingangsElement();            /** Lesen des privaten ↩
       Attributes isEingangsElement eines einzelnen Objekts vom Typ ↩
```

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

A.17 SchaltwerkElement.cpp

```
1  // SchaltwerkElement.cpp
2  //
3  //
4  //
5
6  #include "SchaltwerkElement.h"
7
8
9    /** Konstruktor der Klasse. Er soll beim Anlegen der Klasse alle ↩
       Attribute mit dem Wert 0 bzw Null fÃ¼r Zeiger initialisiern. ↩
       Ausserdem
10   bekommt der Konstruktor einen Zeiger auf ein Element der ↩
       Bibliotheksdatenbank und speichert es in das Attribut typ.*/
11   SchaltwerkElement::SchaltwerkElement(  GatterTyp* gTyp ){
12     name = "";
13     typ = gTyp;
14     laufzeitEinzelgatter = 0;
15     nachfolgerElemente[0] = NULL; //schÃ¶ner lÃ¶sen?
16     nachfolgerElemente[1] = NULL;
17     nachfolgerElemente[2] = NULL;
18     nachfolgerElemente[3] = NULL;
19     nachfolgerElemente[4] = NULL;
20     anzahlNachfolger = 0;
21     isEingangsElement = false;
22     isAusgangsElement = false;
23     anzahlEingangssignale = 0;
24   }
25
26
27   SchaltwerkElement::~SchaltwerkElement() { }          /** Destruktor ↩
       der Klasse.*/
28
29   /** Die folgenden Methoden dienen zum Lesen der privaten Attribute ↩
       eines einzelnen Objekts vom Typ
30     SchaltwerkElement.*/
31
32   string SchaltwerkElement::getName(){                 /** Lesen des ↩
       privaten Attributes name eines einzelnen Objekts vom Typ ↩
       SchaltwerkElement. */
33     return name;
34   }
35
36   GatterTyp* SchaltwerkElement::getTyp(){              /** Lesen des ↩
       privaten Attributes typ eines einzelnen Objekts vom Typ ↩
       SchaltwerkElement. */
37     return typ;
38   }
39
40   double SchaltwerkElement::getLaufzeitEinzelgatter(){  /** Lesen des ↩
       privaten Attributes laufzeitEinzelgatter eines einzelnen Objekts↩
        vom Typ SchaltwerkElement. */
41     return laufzeitEinzelgatter;
42   }
43
44   SchaltwerkElement* SchaltwerkElement::getNachfolger( int pos){  /** ↩
       Lesen des privaten Attributes nachfolgerElemente eines einzelnen↩
        Objekts vom Typ SchaltwerkElement. */
```

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

A.18 ListenElement.h

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

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 0
29   ~GraphErzeuger();          /// unnuetzer Destruktor
30
31     void listeAnlegen( SignalListeErzeuger signallist);     /** ↩
         durchlaeuft die SignalListe, weisst jeder Quelle ein ↩
         Schaltwerk zu, uebernimmt Eigenschaften der
32                                                            Signale ↩
                                                                 und ↩
                                                                 Gattertypen↩
                                                                  und ↩
                                                                 verknuepft↩
                                                                  die ↩
                                                                 Schaltwerke↩
                                                                  mit ↩
                                                                 je ↩
                                                                 einem ↩
                                                                 Listenelement↩
                                                                  und ↩
                                                                 die ↩
                                                                 ListenElemente↩

33                                                            untereinander↩
                                                                 */
34     void graphErzeugen( SignalListeErzeuger signallist);    /** ↩
         durchlaeuft oben angelegte Liste und verknuepft auf Grundlage ↩
         der Signalliste die Schaltwerke
35                                                            miteinander↩
                                                                 */
```

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

A.21 GraphErzeuger.cpp

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

```
58      endElement = NULL;
59      signale = NULL;
60      anzahlSignale = 0;
61      gAnzahl = 0;
62
63      short eingaenge = 0;
64      short ausgaenge = 0;
65
66      ListenElement* tmpElement = NULL;
67      anzahlSignale = signallist.getAnzahlSignale();
68
69  /// geht Signalliste durch
70      for (int i = 0; i < anzahlSignale; i++)
71      {
72          Signal tmpSignal = *signallist.getSignal( i );
73
74          if ((tmpSignal.getSignalTyp() == eingang) )          /// prueft↩
                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
89
90                  ListenElement* newListenElement = new ListenElement();
91                  SchaltwerkElement* newSchaltwerkElement = new ↩
                        SchaltwerkElement( tmpGatter );
92
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());
98
99                  /// pruefen ob Ausgang
100                 if ( tmpSignal.getSignalTyp() == ausgang )
101                 {
102                     newSchaltwerkElement->setIsAusgangsElement(true);
103                     debug_msg("INFO: ausgang gefunden");
```

```
104                     ausgaenge++;
105                 }
106
107                 /// verknuepfen von Schaltwerkselement mit Listenelement
108                 newListenElement->setSchaltwerkElement( newSchaltwerkElement );
109                 gAnzahl++;
110                 debug_msg("INFO: "<<gAnzahl<<". ListenElement angelegt vom Typ "<< tmpSignal.getQuellenTyp()<<" !");
111
112
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                 }
120                 else
121                 {
122                     tmpElement->setNextElement( newListenElement );
123                     endElement = newListenElement;
124                 }
125                 tmpElement = newListenElement;
126
127             }
128             else   // von leerer Quelle Abfrage, um ungenutzte Signake zu erkennen
129             {
130                 cout << "Fehler! Unbenutztes Signal gefunden" << endl;
131                 cin.ignore();
132                 cin.get();
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() <<" ist Eingang");
158                     }
159                 }
160             }
161         }
162     }
163
164     /// prueft, ob es unbeschaltete Eingaenge gibt
165     short tmpZaehler ;
166     for (ListenElement* ptr = startElement; ptr != NULL; ptr = ptr->getNextElement())   /// durchlaeuft die Listenelemente
167     {
168         tmpZaehler = 0;
169
170         debug_msg("INFO:-----");
171
172         for (int z = 0; z < signallist.getAnzahlSignale(); z++)                 ///danach die Signalliste
173         {
174             for (int r = 0; r < signallist.getSignal(z)->getAnzahlZiele(); r++)     /// und die Ziele eines jeden Signals
175             {
176
177                 /// prueft, ob das Signalziel mit dem SchaltwerkElementsnamen uebereinstimmt und erhoeht den Eingangszaehler bei Erfolg um 1
178                 if ( signallist.getSignal(z)->getZiel(r) == ptr->getSchaltwerkElement()->getName())
179                 {
180                     tmpZaehler += 1;
181
182                     debug_msg("INFO: "<< tmpZaehler <<". Eingang von "<< ptr->getSchaltwerkElement()->getName() <<" gefunden");
183                 }
184             }
185         }
186         /// falls dff mit clk, hat die Bib einen Eingang zu wenig, wird hier korrigiert
187         if (ptr->getSchaltwerkElement()->getTyp()->getName()=="dff")  // weil der clock eingang nicht mit eingelesen wird.. sollte man vlt noch aendern
188         {
189             tmpZaehler -= 1;
```

```
190         }
191         /// check, ob Schaltwerk und Bib fuer das jeweilige Element ↩
                dieselbe Anzahl Eingaenge verzeichnet haben
192         if (ptr->getSchaltwerkElement()->getTyp()->getEingaenge() != ↩
                tmpZaehler)
193         {
194             cout << "Fehler!\nAnzahl Eingaenge laut Bibliothek: \t"<<↩
                    ptr->getSchaltwerkElement()->getTyp()->getEingaenge()↩
                    <<endl
195             <<"Anzahl Eingaenge laut Schaltwerk: \t"<<tmpZaehler << ↩
                    endl;
196             cin.ignore();
197             cin.clear();
198             cin.get();
199         }
200     }
201
202 }
203
204
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         }
252
253     }
254
255 }
256
257 void GraphErzeuger::listenAusgabe ( )          /// gibt die ↩
    Listenelemente mit Gatternamen und ihre NachfolgeGatter aus
258 {
259
260     for ( ListenElement* ptr = startElement; ptr != NULL; ptr = ptr->↩
        getNextElement())  /// geht die ListenElemente durch
261     {
262
263         cout << "-----------\n"<<endl
264         <<"Gattername:  \t\t" <<  ptr->getSchaltwerkElement()->getName↩
            () <<endl                    /// Gattername
265         <<"Gattertyp:  \t\t" << ptr->getSchaltwerkElement()->getTyp()↩
            ->getName() <<endl;          /// GatterTyp
266
267         /// evtle 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
3
4  #include <map>
5  #include "ListenElement.h"
6  #include "Faktoren.h"
7  #include "GraphErzeuger.h"
8  #include <vector>
9
10 struct DFS_Daten
11 {
12     SchaltwerkElement* VaterElement;
13     double PfadLaufzeit;
14 };
15
16 class LaufzeitAnalysator
17 {
18 private:
19
20     Faktoren* faktoren;
21     GraphErzeuger* gE;
22
23     long frequenz;
24     string uebergangspfad;
25     string ausgangspfad;
26     double laufzeitUebergangspfad;
27     double laufzeitAusgangspfad;
28     bool zyklusFound;
29     bool zyklensuche(SchaltwerkElement* se);
30     void DFS(ListenElement* s);
31
32     map < SchaltwerkElement* , DFS_Daten > DFS_Zwischenspeicher;
33
34     void DFS_Visit(SchaltwerkElement* k, SchaltwerkElement* s);
35
36
37 public:
38     LaufzeitAnalysator(GraphErzeuger* gE, Faktoren* f);
39     virtual ~LaufzeitAnalysator();
40
41     void berechne_LaufzeitEinzelgatter();
42
43     bool DFS_startSuche(GraphErzeuger* ge);
44     double maxFrequenz(long freq);
45
46 protected:
47
48 };
49
50 #endif // LAUFZEITANALYSATOR_H
```

A.23 LaufzeitAnalysator.cpp

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

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

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

```
144                 //cout << "Ã¼bergangspfad: "<<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                 }
150
151             }
152         }
153
154         else if (DFS_Zwischenspeicher[v].PfadLaufzeit < (↩
                DFS_Zwischenspeicher[k].PfadLaufzeit +k->↩
                getLaufzeitEinzelgatter()))
155         {
156
157             if( ( ( DFS_Zwischenspeicher[ v ].PfadLaufzeit != 0 ) or (↩
                    v== s ) ) and ( DFS_Zwischenspeicher[ v ].↩
                   VaterElement != k) )
158             {
159
160                 DFS_Zwischenspeicher[v].VaterElement =k;
161
162
163                 if(zyklensuche(v))
164                 {
165                     zyklusFound = true;
166                     cout << "Fehler Zyklensuche!"<<endl;
167
168                 }
169             }
170             DFS_Zwischenspeicher[v].PfadLaufzeit = ↩
                   DFS_Zwischenspeicher[k].PfadLaufzeit + k->↩
                   getLaufzeitEinzelgatter();
171             DFS_Zwischenspeicher[v].VaterElement = k;
172
173             DFS_Visit(v,s);
174
175         }
176
177     }
178
179     if(k->getIsAusgangsElement() and (laufzeitAusgangspfad < (↩
           DFS_Zwischenspeicher[k].PfadLaufzeit + k->↩
           getLaufzeitEinzelgatter())))
180     {
181         laufzeitAusgangspfad  = DFS_Zwischenspeicher[k].PfadLaufzeit +↩
                k->getLaufzeitEinzelgatter();
182
183         ausgangspfad = k->getName();
184
185         for(SchaltwerkElement * j = k; j != s; j= DFS_Zwischenspeicher↩
               [j].VaterElement)
186         {
187             ausgangspfad.insert(0,(DFS_Zwischenspeicher[j].↩
                   VaterElement->getName() + "->"));
188         }
```

```
189
190     }
191
192 }
193
194 double LaufzeitAnalysator::maxFrequenz(long freq)
195 {
196
197     cout << "LÃ¤ngster Pfad im Ã¼berfuehrungsschaltnetz:" << endl;
198     cout << uebergangspfad << endl << endl;
199     cout << "Maximale Laufzeit der Pfade im Ã¼berfuehrungsschaltnetz: "
        << laufzeitUebergangspfad << " ps" << endl;
200     cout << endl;
201     cout << "LÃ¤ngster Pfad im Ausgangsschaltnetz:" << endl;
202     cout << ausgangspfad << endl << endl;
203     cout << "Maximale Laufzeit der Pfade im Ausgangsschaltnetz: " <<
        laufzeitAusgangspfad << " ps" << endl;
204     cout << endl;
205     cout << "-------------------------------------------------"<<endl;
206     long double maxF = 1/ (dynamic_cast<Flipflop*>(( gE->getBibliothek
        ()->getBibElement("dff")))->getSetupTime() * 0.000000000001 +
        laufzeitUebergangspfad * 0.000000000001 );
207
208     if(maxF>1000){
209         if(maxF>1000000){
210
211         cout << "maxFrequenz: "<<maxF/1000000<<" MHz"<<endl;
212         }
213         else{
214             cout << "maxFrequenz: "<<maxF/1000<<" kHz"<<endl;
215         }
216     }
217
218     cout << "-------------------------------------------------"<<endl;
219     if(maxF < freq){
220         cout << "Frequenz zu gross"<<endl;
221     }
222     else{
223         cout << "Frequenz okay"<<endl;
224
225     }
226
227
228 }
```

A.24 cross-compatibility.h

```
1  // Funktionen um Windows und Linux KompatibilitÃ¤t zu ermÃ¶glichen
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 kÃ¶nnen
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 }*/
```