

# Source code - table of contents

<b>3 Source code</b>	<b>8</b>
3.1 Changes to supplied source code . . . . .	8
3.1.1 devices . . . . .	8
3.1.2 network . . . . .	10
3.1.3 logsim . . . . .	12
3.1.4 monitors . . . . .	13
3.2 GUI . . . . .	17
3.2.1 Helper class: circuit . . . . .	17
3.2.2 Helper class: observers . . . . .	21
3.2.3 Main GUI: dialogs and frame . . . . .	22
3.2.4 Main GUI: monitor traces canvas . . . . .	28
3.2.5 GUI: miscellaneous widgets and functions . . . . .	34
3.2.6 GUI: widget IDs . . . . .	38
3.2.7 Device editing GUI: dialogs . . . . .	38
3.2.8 Device editing GUI: information panels . . . . .	43

## 3 Source code

### 3.1 Changes to supplied source code

Where appropriate, changes to the supplied source code are shown as diffs, and summaries of the changes are given. For files where many parts have changed and a diff would be hard to read, the entire new file is shown.

I wrote all the changes to the supplied source code listed here, except for the changes to `logsim.cc` and `logsim.h`, which were performed partly by other team members.

#### 3.1.1 devices

**Summary:** Three functions added, mainly for use by the device editing part of the GUI. Small modification to `updateclocks()` so that changing **frequency** after running the simulation can no longer cause a clock to stop changing state. `maxmachinecycles` is now dynamically set depending on the number of devices, to make it less likely that simulation will fail unnecessarily. It will still fail if an inverter is connected to itself, but will no longer fail for 20 inverters connected in series.

Listing 1: devices.h diff

```
@@ -55,6 +55,15 @@ class devices {
    void debug (bool on);
    /* Used to set debugging switch.                                     */

+ // Changes the number of inputs for a gate
+ void SetGateInputCount(devlink d, int newCount);
+
+ // Gets the name for an input of a gate (n=1 -> 'I1', n=2 -> 'I2', etc)
+ name GetGateInputName(int n);
+
+ // Checks whether all inputs are connected
+ bool CheckDeviceInputs(devlink d);
+
    devices (names* names_mod, network* net_mod);
    /* Called to initialise module.                                     */
};
```

Listing 2: devices.cc diff

```

@@ -125,14 +125,7 @@ void devices::makegate (devicekind dkind, name did, int
ninputs, bool& ok)
    netz->adddevice (dkind, did, d);
    netz->addoutput (d, blankname);
    for (n = 1; n <= ninputs; n++) {
-       iname = "I";
-       if (n < 10) {
-           iname += ((char) n) + '0';
-       } else {
-           iname += ((char) (n / 10)) + '0';
-           iname += ((char) (n % 10)) + '0';
-       }
-       netz->addinput (d, nmz->lookup (iname));
+       netz->addinput(d, GetGateInputName(n));
    }
}
}
@@ -342,7 +335,7 @@ void devices::updateclocks (void)
    devlink d;
    for (d = netz->devicelist (); d != NULL; d = d->next) {
        if (d->kind == aclock) {
-           if (d->counter == d->frequency) {
+           if (d->counter >= d->frequency) {
                d->counter = 0;
                if (d->olist->sig == high)
                    d->olist->sig = falling;
@@ -364,7 +357,8 @@ void devices::updateclocks (void)
    */
    void devices::executedevices (bool& ok)
    {
-       const int maxmachinecycles = 20;
+       int maxmachinecycles = 20;
+       int count = 0;
        devlink d;
        int machinecycle;
        if (debugging)
@@ -387,9 +381,11 @@ void devices::executedevices (bool& ok)
            case xorgate:  execxorgate (d);          break;
            case dtype:    execdtype (d);            break;
        }
+       if (machinecycle==1) count++;
        if (debugging)
            showdevice (d);
    }
+   if (machinecycle==1) maxmachinecycles = 20 + 2*count;
} while ((! steadystate) && (machinecycle < maxmachinecycles));

```

```

    if (debugging)
        cout << "End of execution cycle" << endl;
@@ -464,4 +460,48 @@ void devices::debug (bool on)
    qbarpin = nmz->lookup("QBAR");
}

+name devices::GetGateInputName(int n)
+{
+ // moved from makegate() into new function
+ namestring iname = "I";
+ if (n < 10) {
+     iname += ((char) n) + '0';
+ } else {
+     iname += ((char) (n / 10)) + '0';
+     iname += ((char) (n % 10)) + '0';
+ }
+ return nmz->lookup(iname);
+}
+
+void devices::SetGateInputCount(devlink d, int newCount)
+{
+ if (!d) return;
+ int oldCount = GetLinkedListLength(d->ilist);
+ if (newCount<oldCount)
+ {
+     for (int i=0; i<oldCount-newCount; i++)
+     {
+         inplink nextI = d->ilist->next;
+         delete d->ilist;
+         d->ilist = nextI;
+     }
+ }
+ else
+ {
+     for (int i=oldCount+1; i<=newCount; i++)
+     {
+         netz->addinput(d, GetGateInputName(i));
+     }
+ }
+}

+bool devices::CheckDeviceInputs(devlink d)
+{
+ inplink i = d->ilist;
+ while (i != NULL)
+ {

```

```

+   if (i->connect == NULL) return false;
+   i = i->next;
+ }
+ return true;
+}

```

### 3.1.2 network

**Summary:** Functions added for getting linked list length, deleting or disconnecting a device, and obtaining device and input/output name strings. `outputrec` now has a pointer to the parent device, to make it easier for the device editing part of the GUI to find which device an output is connected to.

Listing 3: network.h diff

```

@@ -2,6 +2,9 @@
#define network_h

#include "names.h"
+ #include <string>
+
+ using namespace std;

/* Network specification */

@@ -10,10 +13,12 @@
     norgate, xorgate, dtype, baddevice
 } devicekind;

+ struct devicerec;
struct outputrec {
    name      id;
    asignal   sig;
    outputrec* next;
+ devicerec* dev;
};
typedef outputrec* outlink;
struct inputrec {
@@ -36,6 +41,18 @@ struct devicerec {
};
typedef devicerec* devlink;

+ template <class T>
+ int GetLinkedListLength(T item)

```

```

+{
+ int count = 0;
+ while (item != NULL)
+ {
+     count++;
+     item = item->next;
+ }
+ return count;
+}
+
+ class network {
+     names* nmz; // the instantiation of the names class that we are going to
+                 use.

@@ -72,12 +89,23 @@ class network {
/* 'outp' output of device 'odev'. 'ok' is set true if operation    */
/* succeeds.                                                         */

- void checknetwork (bool& ok);
+ void checknetwork (bool& ok, bool silent=false);
/* Checks that all inputs are connected to an output.              */

    network (names* names_mod);
/* Called on system initialisation.                                */

+ string getsignalstring(name dev, name p=blankname);
+ string getsignalstring(devlink dev, outlink o);
+ string getsignalstring(devlink dev, inlink i);
+ /* Returns the string corresponding to the given device and pin    */
+
+ // Disconnects all inputs connected to the given output
+ void disconnectoutput(outlink o);
+
+ // Deletes a device (after disconnecting the outputs)
+ void deletedevice(devlink dTarget);
+
private:
    devlink devs;           // the list of devices
    devlink lastdev;        // last device in list of devices

```

Listing 4: network.cc diff

```

@@ -133,6 +133,7 @@ void network::addinput (devlink dev, name iid)
void network::addoutput (devlink dev, name oid)
{
    outlink o = new outputrec;
+ o->dev = dev;

```

```

o->id = oid;
o->sig = low;
o->next = dev->olist;
@@ -170,14 +171,17 @@ void network::makeconnection (name idev, name inp, name
    odev, name outp, bool& o
    * Checks that all inputs are connected to an output.
    *
    */
-void network::checknetwork (bool& ok)
+void network::checknetwork (bool& ok, bool silent)
{
    devlink d;
    inplink i;
    ok = true;
    for (d = devs; d != NULL; d = d->next)
        for (i = d->ilist; i != NULL; i = i->next)
-        if (i->connect == NULL) {
+        if (i->connect == NULL)
+        {
+            if (!silent)
+            {
                cout << "Unconnected Input : ";
                nmz->writename (d->id);
                if (i->id != blankname) {
@@ -185,6 +189,7 @@ void network::checknetwork (bool& ok)
                    nmz->writename (i->id);
                }
                cout << endl;
+            }
+            ok = false;
        }
    }
@@ -205,3 +210,87 @@ void network::checknetwork (bool& ok)
    lastdev = NULL;
}

+// Delete a device, after disconnecting it. Monitors should be deleted too,
+    but this function only handles deleting the devicerec and inputs+outputs
+    (use circuit::RemoveDevice instead).
+void network::deletedevice(devlink dTarget)
+{
+    if (!dTarget) return;
+    devlink d = devicelist();
+    devlink dPrev = NULL;
+    while (d!=NULL)
+    {

```

```

        if (d == dTarget)
        {
            outplink o = d->olist, oNext;
            while (o != NULL)
            {
                disconnectoutput(o);
                oNext = o->next;
                delete o;
                o = oNext;
            }
            inplink i = d->ilist, iNext;
            while (i != NULL)
            {
                iNext = i->next;
                delete i;
                i = iNext;
            }
            if (devs == d)
                devs = d->next;
            else
                dPrev->next = d->next;
            if (lastdev == d)
                lastdev = dPrev;
            break;
        }
        dPrev = d;
        d = d->next;
    }
+}
+
+// Disconnects all inputs connected to the given output
+void network::disconnectoutput(outplink o)
+{
+    devlink d = devicelist();
+    while (d!=NULL)
+    {
+        inplink i = d->ilist;
+        while (i!=NULL)
+        {
+            if (i->connect == o)
+                i->connect = NULL;
+            i = i->next;
+        }
+        d = d->next;
+    }
+}
+}

```

```

+
+string network::getsignalstring(name dev, name p)
+{
+ if (dev==blankname)
+   return "";
+ string str = nmz->getnamestring(dev);
+ if (p != blankname)
+   str += "." + nmz->getnamestring(p);
+ return str;
+}
+
+string network::getsignalstring(devlink d, outlink o)
+{
+ if (d==NULL)
+   return "";
+ name opn = blankname;
+ if (o==NULL)
+   return getsignalstring(d->id);
+ return getsignalstring(d->id, o->id);
+}
+
+string network::getsignalstring(devlink d, inlink i)
+{
+ if (d==NULL)
+   return "";
+ name opn = blankname;
+ if (i==NULL)
+   return getsignalstring(d->id);
+ return getsignalstring(d->id, i->id);
+}

```

### 3.1.3 logsim

**Summary:** Command line argument is now optional for the GUI, and a MyFrame member function is used to load the file if using the GUI. Additional class `error` added to count and display errors which occur during parsing.

Listing 5: logsim.h diff

```

@@ -7,6 +7,7 @@
#include "monitor.h"
#include "network.h"
#include "parser.h"
+#include "scanner.h"

```

```

class MyApp: public wxApp
{
@@ -19,6 +20,8 @@ class MyApp: public wxApp
    monitor *mmz; // pointer to the monitor class
    scanner *smz; // pointer to the scanner class
    parser *pmz; // pointer to the parser class
+ error *erz; // pointer to the error class
+ bool ok;
};

#endif /* logsim_h */

```

Listing 6: logsim.cc

```

1 #include "logsim.h"
2 #include "userint.h"
3 #include "gui.h"
4 #include <GL/glut.h>
5
6 #define USE_GUI
7
8 IMPLEMENT_APP(MyApp)
9
10 bool MyApp::OnInit()
11 // This function is automatically called when the application starts
12 {
13 #ifndef USE_GUI
14   if (argc != 2) { // check we have one command line argument
15     wcout << "Usage:      " << argv[0] << " [filename]" << endl;
16     exit(1);
17   }
18 #endif
19
20   // Construct the six classes required by the innards of the logic
   simulator
21   nmz = new names();
22   netz = new network(nmz);
23   dmz = new devices(nmz, netz);
24   mmz = new monitor(nmz, netz);
25
26 #ifndef USE_GUI
27   // glutInit cannot cope with Unicode command line arguments, so we
   pass
28   // it some fake ASCII ones instead
29   char **tmp1; int tmp2 = 0; glutInit(&tmp2, tmp1);
30   // Construct the GUI
31   MyFrame *frame = new MyFrame(NULL, wxT("Logic simulator"),
     wxDefaultPosition, wxSize(800, 600), nmz, dmz, mmz, netz);
32   frame->Show(true);

```

```

33 if (argc == 2)
34 {
35     frame->loadFile(wxString(argv[1]).mb_str());
36 }
37 return(true); // enter the GUI event loop
38 #else
39 smz = new scanner(nmz, wxString(argv[1]).mb_str(), ok);
40 if (!ok)
41 {
42     return(false);
43 }
44 erz = new error(smz);
45 pmz = new parser(netz, dmz, mmz, smz, erz);
46 if (pmz->readin ()) { // check the logic file parsed correctly
47     // Construct the text-based interface
48     userint umz(nmz, dmz, mmz);
49     umz.userinterface();
50 }
51 return(false); // exit the application
52 #endif /* USE_GUI */
53 }

```

### 3.1.4 monitors

**Summary:** Converted to use STL vectors, and to use pointers to the device and output objects instead of the name for each. New functions getsamplecount, getsignalstring, IsMonitored added.

Listing 7: monitor.h diff

```

@@ -4,27 +4,28 @@
#include "names.h"
#include "network.h"
#include "devices.h"
+ #include <vector>
+ #include <string>
+ using namespace std;

const int maxmonitors = 10;    /* max number of monitor points */
const int maxcycles = 50;    /* max number of cycles per run */

+ typedef vector<asignal> signaltrace;
+
struct moninfo {
- name devid;
+ devlink d;

```

```

outlink op;
+ signaltrace disp;
};
- struct monitortable {
- int used;
- moninfo sigs[maxmonitors];
- };
- typedef asignal signaltrace[maxcycles];
+ typedef vector<moninfo> montable;

class monitor {
names* nmz;    // version of names class to use.
network* netz; // version of the network class to use.

- monitortable mtab;                // table of monitored signals
+ montable mtab;                    // table of monitored signals
int cycles;                        // counts clock cycles
- signaltrace disp[maxmonitors];

public:
void makemonitor (name dev, name outp, bool& ok);
@@ -60,6 +61,16 @@ class monitor {

monitor (names* names_mod, network* network_mod);
/* Called to initialise the monitor module. */

+
+ // Returns the name of the monitored signal as a string
+ string getsignalstring(int m);
+
+ // Returns the number of samples recorded by the n'th monitor
+ int getsamplecount(int m);
+
+ // Returns true if the given output is being monitored
+ bool IsMonitored(outlink o);
+
};

#endif /* monitor_h */

```

Listing 8: monitor.cc

```

1 #include <iostream>
2 #include "monitor.h"
3
4 using namespace std;
5

```

```

6  /*
   *****
7  *
8  * Sets a monitor on the 'outp' output of device 'dev' by placing an
9  * entry in the monitor table. 'ok' is set true if operation succeeds
10  *
11  */
12 void monitor::makemonitor (name dev, name outp, bool& ok)
13 {
14     devlink d;
15     outplink o;
16     d = netz->finddevice(dev);
17     ok = (d != NULL);
18     if (ok)
19     {
20         o = netz->findoutput(d, outp);
21         ok = (o != NULL);
22         if (ok)
23         {
24             moninfo mon;
25             mon.d = d;
26             mon.op = o;
27             mtab.push_back(mon);
28         }
29     }
30 }
31
32 /*
33 *****
34 *
35 * Removes the monitor set on the 'outp' output of device 'dev'. 'ok'
36 * is
37 * set true if operation succeeds.
38 *
39 */
40 void monitor::remmonitor (name dev, name outp, bool& ok)
41 {
42     bool found;
43     found = false;
44     for (montable::iterator it=mtab.begin(); it!=mtab.end(); ++it)
45     {
46         if ((it->d->id == dev) && (it->op->id == outp))
47         {
48             found = true;
49             mtab.erase(it);
50             break;
51         }
52     }

```

```

51     }
52     ok = found;
53 }
54
55 /*
56 *****
57 *
58 * Returns number of signals currently monitored.
59 *
60 */
61 int monitor::moncount (void)
62 {
63     return mtab.size();
64 }
65
66 /*
67 *****
68 *
69 * Returns signal level of n'th monitor point.
70 *
71 */
72 signal monitor::getmonsignal (int n)
73 {
74     return (mtab[n].op->sig);
75 }
76
77 /*
78 *****
79 *
80 * Returns name of n'th monitor.
81 *
82 */
83 void monitor::getmonname (int n, name& dev, name& outp)
84 {
85     dev = mtab[n].d->id;
86     outp = mtab[n].op->id;
87 }
88
89 /*
90 *****
91 *
92 * Initialises monitor memory in preparation for a new output
   sequence.

```

```

93  *
94  */
95  void monitor::resetmonitor (void)
96  {
97      int n;
98      for (n = 0; n < moncount (); n++)
99          mtab[n].disp.clear();
100      cycles = 0;
101  }
102
103  /*
104  *****
105  *
106  * Called every clock cycle to record the state of each monitored
107  * signal.
108  *
109  */
110  void monitor::recordsignals (void)
111  {
112      int n;
113      for (n = 0; n < moncount (); n++)
114          mtab[n].disp.push_back(getmonsignal(n));
115      cycles++;
116  }
117
118  /*
119  *****
120  *
121  * Access recorded signal trace, returns false if invalid monitor
122  * or cycle.
123  */
124  bool monitor::getsignaltrace(int m, int c, asignal &s)
125  {
126      if ((c < cycles) && (m < moncount ()) && c < mtab[m].disp.size()) {
127          s = mtab[m].disp[c];
128          return true;
129      }
130      return false;
131  }
132
133  /*
134  *****
135  *
136  * Get number of recorded cycles for the monitor, returns -1 if
137  * invalid
138  * monitor.

```

```

137  *
138  */
139  int monitor::getsamplecount(int m)
140  {
141      if (m < moncount ()) {
142          return mtab[m].disp.size();
143      }
144      return -1;
145  }
146
147  /*
148  *****
149  *
150  * Displays state of monitored signals.
151  */
152  void monitor::displaysignals (void)
153  {
154      const int margin = 20;
155      int n, i;
156      name dev, outp;
157      int namesize;
158      for (n = 0; n < moncount (); n++) {
159          getmonname (n, dev, outp);
160          namesize = nmz->namelength (dev);
161          nmz->writename (dev);
162          if (outp != blankname) {
163              cout << ".";
164              nmz->writename (outp);
165              namesize = namesize + nmz->namelength (outp) + 1;
166          }
167          if ((margin - namesize) > 0) {
168              for (i = 0; i < (margin - namesize - 1); i++)
169                  cout << " ";
170              cout << ".";
171          }
172          for (i = 0; i < cycles; i++)
173              switch (mtab[n].disp[i]) {
174                  case high:    cout << "-"; break;
175                  case low:     cout << "_"; break;
176                  case rising:  cout << "/"; break;
177                  case falling: cout << "\\"; break;
178              }
179          cout << endl;
180      }
181  }
182
183  /*
184  *****

```



```

185  *
186  * Called to initialise the monitor module.
187  * Remember the names of the shared names and network modules.
188  *
189  */
190  monitor::monitor (names* names_mod, network* network_mod)
191  {
192      nmz = names_mod;
193      netz = network_mod;
194  }
195
196  // Returns the name of the monitored signal as a string
197  string monitor::getsignalstring(int m)
198  {
199      if (m<0 || m>=moncount()) return "";
200      return netz->getsignalstring(mtab[m].d->id, mtab[m].op->id);
201  }
202
203  bool monitor::IsMonitored(outplink o)
204  {
205      for (int n = 0; n < moncount (); n++)
206      {
207          if (mtab[n].op == o) return true;
208      }
209      return false;
210  }

```

## 3.2 GUI

I was responsible for the whole of the GUI, except for some contributions by other team members to the `MyFrame::loadFile()` function.

### 3.2.1 Helper class: circuit

Listing 9: circuit.h

```
1 #ifndef circuit_h
2 #define circuit_h
3
4 #include "names.h"
5 #include "network.h"
6 #include "devices.h"
7 #include "monitor.h"
8 #include "observer.h"
9 #include <vector>
10 using namespace std;
11
12 class circuit;
13
14
15 // Many bits of the GUI involve manipulating (creating, searching,
16 // sorting) vectors where each element needs to contain one or more
17 // pointers to circuit elements, with an associated string. This
18 // class is a unified way of doing that for all circuit elements (
19 // inputs, outputs, and devices).
20
21 class CircuitElementInfo
22 {
23 public:
24     devlink d;
25     outlink o;
26     inlink i;
27     string namestr;
28     CircuitElementInfo() : d(NULL), o(NULL), i(NULL), namestr("") {}
29     CircuitElementInfo(devlink dev, string str="") : d(dev), o(NULL), i
30     (NULL), namestr(str) {}
31     CircuitElementInfo(devlink dev, outlink outp, string str="") : d(
32     dev), o(outp), i(NULL), namestr(str) {}
33     CircuitElementInfo(outlink outp, string str="") : d(outp->dev), o(
34     outp), i(NULL), namestr(str) {}
35     CircuitElementInfo(devlink dev, inlink inp, string str="") : d(dev
36     ), o(NULL), i(inp), namestr(str) {}
37 };
38
39 class CircuitElementInfoVector : public vector<CircuitElementInfo>
40 {
```

```
32 public:
33     // push_back() all devices in a linked list of devices, all outputs
34     // in a linked list of devices, or all inputs in a linked list of
35     // devices
36     void push_back_all_devs(devlink d);
37     void push_back_all_outputs(devlink d);
38     void push_back_all_inputs(devlink d);
39     // push_back() all outputs or inputs for a particular device
40     void push_back_dev_outputs(devlink d);
41     void push_back_dev_inputs(devlink d);
42     // Sets namestr for each element to the device or signal name (e.g.
43     // S1, or G1.I1, or DT1.CLK)
44     void UpdateSignalNames(circuit* c);
45 private:
46     template <class T> void push_back_iolist(devlink d, T item);
47 };
48
49 bool CircuitElementInfo_namestrcmp(const CircuitElementInfo a, const
50 CircuitElementInfo b); // to alphabetically sort a vector<
51 outputinfo>
52 bool CircuitElementInfo_iconnect_namestrcmp(const CircuitElementInfo
53 a, const CircuitElementInfo b); // to sort a vector<outputinfo> by
54 input connected state then alphabetically by namestr
55
56 // Used in the device editing GUI to store a pointer to the currently
57 // selected device and notify widgets when it changes. This is here
58 // instead of in a gui file because it contains nothing specific to
59 // wxWidgets and may be useful elsewhere
60
61 class SelectedDevice
62 {
63 private:
64     devlink selected;
65 public:
66     ObserverSubject changed;
67     devlink Get()
68     {
69         return selected;
70     }
71     void Set(devlink d)
72     {
73         if (d!=selected)
74         {
75             selected = d;
76             changed.Trigger();
77         }
78     }
79 };
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
```

```

72 // This class is a more convenient way of passing around a set of
    names, monitors, devices, and network modules.
73 // It also handles observers for changes to the circuit.
74 // It provides a home for a few functions that are more closely
    related to the circuit itself than to the GUI or MyFrame, such as
    running the network
75 class circuit
76 {
77 private:
78     // Module pointers
79     names *names_mod;
80     devices *devices_mod;
81     monitor *monitor_mod;
82     network *network_mod;
83     bool ownModules;
84     void AllocModules();
85     void DestroyModules();
86
87     int continuedCycles;// how many simulation cycles were completed
        last time the run or continue button was used
88     int totalCycles;// how many simulation cycles have been completed
        in total
89
90 public:
91     circuit();
92     circuit(names *existing_names_mod, devices *existing_devices_mod,
        monitor *existing_monitor_mod, network *existing_network_mod);
93     virtual ~circuit();
94     // Swap module pointers without changing observers
95     void SwapModules(circuit& source);
96     // clear the circuit, currently done by allocating new modules
97     void Clear();
98     // Clear recorded samples
99     void ResetMonitors();
100    // Simulate the logic circuit, optionally clearing monitors
        beforehand
101    bool Simulate(int ncycles, bool resetBefore=true);
102    // GetUnmonitoredOutputs returns true if there are unmonitored
        outputs in the circuit
103    // unmonitoredOutputsRet is an optional pointer to a vector to
        hold the list of unmonitored outputs
104    bool GetUnmonitoredOutputs(CircuitElementInfoVector *
        unmonitoredOutputsRet=NULL);
105
106    ObserverSubject monitorsChanged;// addition or removal of monitors
107    ObserverSubject circuitChanged;// changes to devices, device
        properties, and connections
108    ObserverSubject monitorSamplesChanged;// simulation has been run or
        continued, or monitor samples have been cleared
109
110    names* nmz() { return names_mod; }

```

```

111 devices* dmz() { return devices_mod; }
112 monitor* mmz() { return monitor_mod; }
113 network* netz() { return network_mod; }
114 int GetTotalCycles() { return totalCycles; }
115 int GetContinuedCycles() { return continuedCycles; }
116 static bool IsDeviceNameValid(string devname);
117 void RemoveDevice(devlink d);
118 };
119
120 #endif

```

Listing 10: circuit.cc

```

1 #include "circuit.h"
2 #include <algorithm>
3 using namespace std;
4
5 void CircuitElementInfoVector::push_back_all_devs(devlink d)
6 {
7     while (d!=NULL)
8     {
9         push_back(CircuitElementInfo(d));
10        d = d->next;
11    }
12 }
13
14 void CircuitElementInfoVector::push_back_all_outputs(devlink d)
15 {
16     while (d!=NULL)
17     {
18         push_back_dev_outputs(d);
19         d = d->next;
20     }
21 }
22
23 void CircuitElementInfoVector::push_back_all_inputs(devlink d)
24 {
25     while (d!=NULL)
26     {
27         push_back_dev_inputs(d);
28         d = d->next;
29     }
30 }
31
32 void CircuitElementInfoVector::push_back_dev_outputs(devlink d)
33 {
34     push_back_iolist(d, d->olist);
35 }
36
37 void CircuitElementInfoVector::push_back_dev_inputs(devlink d)
38 {

```

```

39   push_back_iolist(d, d->iolist);
40 }
41
42 template <class T>
43 void CircuitElementInfoVector::push_back_iolist(devlink d, T item)
44 {
45     while (item != NULL)
46     {
47         push_back(CircuitElementInfo(d, item));
48         item = item->next;
49     }
50 }
51
52 void CircuitElementInfoVector::UpdateSignalNames(circuit* c)
53 {
54     for (CircuitElementInfoVector::iterator it=begin(); it<end(); it++)
55     {
56         if (!it->d)
57         {
58             it->namestr = "";
59         }
60         else
61         {
62             if (it->i)
63                 it->namestr = c->netz()->getsignalstring(it->d,it->i);
64             else if (it->o)
65                 it->namestr = c->netz()->getsignalstring(it->d,it->o);
66             else
67                 it->namestr = c->nmz()->getnamestring(it->d->id);
68         }
69     }
70 }
71
72
73 circuit::circuit()
74 {
75     AllocModules();
76     totalCycles = continuedCycles = 0;
77 }
78
79 circuit::circuit(names *existing_names_mod, devices *
80     existing_devices_mod, monitor *existing_monitor_mod, network *
81     existing_network_mod)
82 {
83     names_mod = existing_names_mod;
84     devices_mod = existing_devices_mod;
85     monitor_mod = existing_monitor_mod;
86     network_mod = existing_network_mod;
87     ownModules = false;
88     totalCycles = continuedCycles = 0;

```

```

88 }
89
90 circuit::~circuit()
91 {
92     DestroyModules();
93 }
94
95 void circuit::AllocModules()
96 {
97     names_mod = new names();
98     network_mod = new network(names_mod);
99     devices_mod = new devices(names_mod, network_mod);
100     monitor_mod = new monitor(names_mod, network_mod);
101     ownModules = true;
102 }
103
104 void circuit::DestroyModules()
105 {
106     if (ownModules)
107     {
108         delete monitor_mod;
109         delete devices_mod;
110         delete network_mod;
111         delete names_mod;
112     }
113     monitor_mod = NULL;
114     devices_mod = NULL;
115     network_mod = NULL;
116     names_mod = NULL;
117     ownModules = false;
118 }
119
120 void circuit::Clear()
121 {
122     DestroyModules();
123     AllocModules();
124     totalCycles = continuedCycles = 0;
125     circuitChanged.Trigger();
126     monitorsChanged.Trigger();
127     monitorSamplesChanged.Trigger();
128 }
129
130 void circuit::SwapModules(circuit& source)
131 {
132     swap(names_mod, source.names_mod);
133     swap(devices_mod, source.devices_mod);
134     swap(monitor_mod, source.monitor_mod);
135     swap(network_mod, source.network_mod);
136     swap(ownModules, source.ownModules);
137 }
138

```

```

139 void circuit::ResetMonitors()
140 {
141     mmz()->resetmonitor();
142     totalCycles = continuedCycles = 0;
143     monitorSamplesChanged.Trigger();
144 }
145
146 bool circuit::Simulate(int ncycles, bool resetBefore)
147 {
148     if (resetBefore)
149     {
150         mmz()->resetmonitor();
151         totalCycles = 0;
152     }
153     continuedCycles = 0;
154
155     // Function to run the network
156     bool ok = true;
157     int n = ncycles;
158
159     while (n > 0 && ok)
160     {
161         dmz()->executedevices(ok);
162         if (ok)
163         {
164             n--;
165             totalCycles++;
166             continuedCycles++;
167             mmz()->recordsignals();
168         }
169         else
170         {
171             cout << "Error: network is oscillating" << endl;
172             ok = false;
173         }
174     }
175
176     monitorSamplesChanged.Trigger();
177     return ok;
178 }
179
180 bool CircuitElementInfo_namestrcmp(const CircuitElementInfo a, const
181     CircuitElementInfo b)
182 {
183     return (a.namestr<b.namestr);
184 }
185
186 bool CircuitElementInfo_iconnect_namestrcmp(const CircuitElementInfo
187     a, const CircuitElementInfo b)
188 {
189     if (a.i && b.i && (a.i->connect!=NULL) != (b.i->connect!=NULL))

```

```

188     return (a.i->connect!=NULL) < (b.i->connect!=NULL);
189     return (a.namestr<b.namestr);
190 }
191
192 bool circuit::GetUnmonitoredOutputs(CircuitElementInfoVector *
193     unmonitoredOutputsRet)
194 {
195     CircuitElementInfoVector unmonitoredOutputs;
196
197     int monCount = mmz()->moncount();
198     devlink d = netz()->devicelist();
199     // Loop through devices
200     while (d!=NULL)
201     {
202         // Loop through device outputs
203         outplink o = d->olist;
204         while (o!=NULL)
205         {
206             // Check whether this output is currently being monitored
207             if (!mmz()->IsMonitored(o))
208             {
209                 unmonitoredOutputs.push_back(CircuitElementInfo(d,o));
210             }
211             o = o->next;
212         }
213         d = d->next;
214     }
215
216     // If a pointer to a vector was supplied, put the list of
217     unmonitored outputs in there
218     if (unmonitoredOutputsRet)
219     {
220         unmonitoredOutputs.UpdateSignalNames(this);
221         unmonitoredOutputsRet->swap(unmonitoredOutputs);
222     }
223
224     // Return true if there are some unmonitored outputs in the circuit
225     return (unmonitoredOutputs.size()>0);
226 }
227
228 bool circuit::IsDeviceNameValid(string devname)
229 {
230     // Checks syntax of a device name string (but not whether a device
231     already exists with that name or if it's a reserved word)
232     if (!devname.length())
233         return false;
234     if (!isalpha(devname[0]))
235         return false;
236     for (string::iterator it=devname.begin(); it<devname.end(); ++it)
237     {
238         if (!isalpha(*it) && !isdigit(*it) && (*it)!='_' && (*it)!='-') return false;

```

```

236 }
237 return true;
238 }
239
240 void circuit::RemoveDevice(devlink d)
241 {
242     if (!d) return;
243
244     // Remove monitors first
245     outlink o = d->olist;
246     bool ok;
247     while (o != NULL)
248     {
249         mmz()->remmonitor(d->id, o->id, ok);
250         o = o->next;
251     }
252
253     // Disconnect, release memory, and remove from linked list of
254     // devices
255     netz()->deletedevice(d);
256
257     circuitChanged.Trigger();
258     monitorsChanged.Trigger();
259 }

```

### 3.2.2 Helper class: observers

Listing 11: observer.h

```

1 #ifndef observer_h
2 #define observer_h
3
4 #include <vector>
5 #include <iostream>
6 using namespace std;
7
8 // Abstract base class for storing callback functions
9 class ObserverCallbackBase
10 {
11 public:
12     virtual void* GetTarget() = 0;
13     virtual void Run() = 0;
14     virtual ~ObserverCallbackBase() {}
15 };
16
17 // Derived class for storing callbacks, using templates to allow
18 // member functions in different classes of the form "void SomeClass
19 // :func()"

```

```

18 // to be called without needing a different derived ObserverCallback
19 // class for each SomeClass
20 template <class T>
21 class ObserverCallback : public ObserverCallbackBase
22 {
23 private:
24     T* targetObject; // pointer to the object which will have a member
25     // function called
26     void (T::*targetFunc)(); // pointer to the member function to call
27 public:
28     ObserverCallback(T* obj, void (T::*func)())
29     {
30         targetObject = obj;
31         targetFunc = func;
32     }
33     virtual void* GetTarget()
34     {
35         // Return the pointer to the target object, used in
36         // ObserverSubject.Detach() to check whether this callback is for
37         // the object being detached
38         return targetObject;
39     }
40     virtual void Run()
41     {
42         // Run the callback function
43         (targetObject->*targetFunc)();
44     }
45 };
46
47 class ObserverSubject
48 {
49 private:
50     vector<ObserverCallbackBase*> callbacks;
51     bool callingFuncs; // Whether Trigger() is currently being run
52 public:
53     ObserverSubject() : callingFuncs(false) {}
54
55     // Register the member function targetFunc of targetObj to be
56     // called when this ObserverSubject is triggered
57     // Does not currently detect and prevent duplicates
58     template <class T>
59     void Attach(T* targetObj, void (T::*targetFunc)())
60     {
61         callbacks.push_back(new ObserverCallback<T>(targetObj, targetFunc
62 ));
63     }
64
65     // Detach all registered callbacks with the object pointer equal to
66     // targetObj
67     void Detach(void* targetObj)
68     {
69

```

```

62     for (vector<ObserverCallbackBase*>::iterator it=callbacks.begin()
63         ; it<callbacks.end(); ++it)
64     {
65         if ((*it)->GetTarget()==targetObj)
66         {
67             delete *it;
68             callbacks.erase(it);
69         }
70     }
71
72     // Runs all registered callback functions
73     void Trigger()
74     {
75         if (callingFuncs)
76         {
77             // Prevent recursion
78             cout << "Error: ObserverSubject::Trigger was called recursively
79             " << endl;
80             return;
81         }
82         callingFuncs = true;
83         for (vector<ObserverCallbackBase*>::iterator it=callbacks.begin()
84             ; it<callbacks.end(); ++it)
85         {
86             (*it)->Run();
87         }
88         callingFuncs = false;
89
90         // Return the number of registered callbacks
91         int CallbackCount()
92         {
93             return callbacks.size();
94         }
95
96         virtual ~ObserverSubject()
97         {
98             for (vector<ObserverCallbackBase*>::iterator it=callbacks.begin()
99                 ; it<callbacks.end(); ++it)
100             {
101                 delete *it;
102             }
103         }
104     };
105
106     // Example usage in observer-test.cc
107 #endif

```

### 3.2.3 Main GUI: dialogs and frame

Listing 12: gui.h

```

1  #ifndef gui_h
2  #define gui_h
3
4  #include "gui-canvas.h"
5  #include <wx/wx.h>
6  #include <wx/spinctrl.h>
7  #include <wx/textctrl.h>
8  #include <wx/listbox.h>
9  #include <wx/scrolwin.h>
10 #include <wx/panel.h>
11 #include "names.h"
12 #include "devices.h"
13 #include "monitor.h"
14 #include "network.h"
15 #include "circuit.h"
16 #include <vector>
17 #include <string>
18
19 using namespace std;
20
21 class MyFrame: public wxFrame
22 {
23 public:
24     MyFrame(wxWindow *parent, const wxString& title, const wxPoint& pos
25         , const wxSize& size,
26         names *names_mod = NULL, devices *devices_mod = NULL,
27         monitor *monitor_mod = NULL, network *net_mod = NULL,
28         long style = wxDEFAULT_FRAME_STYLE); // constructor
29     virtual ~MyFrame();
30     bool loadFile(const char * filename); // loads the given file,
31         returns true if successful
32 private:
33     MyGLCanvas *canvas; // GL drawing area widget
34         to draw traces
35     wxSpinCtrl *spin; // control widget to select
36         the number of cycles
37     wxTextCtrl *outputTextCtrl; // textbox to display
38         messages sent to cout (e.g. error messages from scanner and
39         parser)
40     wxStreamToTextRedirector *outputTextRedirect;
41     SwitchesCheckListBox *switchesCtrl;
42     wxPanel *simctrls_container;
43     wxButton *simctrl_run;
44     wxButton *simctrl_continue;
45     wxButton *monitors_add_btn;
46     wxButton *monitors_rem_btn;
47     circuit* c;

```



```

41 void OnExit(wxCommandEvent& event);    // callback for exit menu
42     item
43 void OnAbout(wxCommandEvent& event);    // callback for about menu
44     item
45 void OnOpenFile(wxCommandEvent& event); // callback for open file
46     menu item
47 void OnButtonRun(wxCommandEvent& event);
48 void OnButtonContinue(wxCommandEvent& event);
49 void OnButtonAddMon(wxCommandEvent& event);
50 void OnButtonDelMon(wxCommandEvent& event);
51 void OnButtonEditDevs(wxCommandEvent& event);
52 void UpdateControlStates();
53 void OnMenuClearCircuit(wxCommandEvent &event);
54
55 DECLARE_EVENT_TABLE()
56 };
57
58 class AddMonitorsDialog: public wxDialog
59 {
60 public:
61     AddMonitorsDialog(circuit* circ, wxWindow* parent, const wxString&
62         title, const wxPoint& pos = wxDefaultPosition, const wxSize& size
63         = wxDefaultSize, long style = wxDEFAULT_DIALOG_STYLE);
64 private:
65     circuit* c;
66     int oldMonCount;
67     CircuitElementInfoVector availableOutputs;
68     wxListBox *lbox;
69     void OnOK(wxCommandEvent& event);
70     DECLARE_EVENT_TABLE()
71 };
72
73 class DelMonitorsDialog: public wxDialog
74 {
75 public:
76     DelMonitorsDialog(circuit* circ, wxWindow* parent, const wxString&
77         title, const wxPoint& pos = wxDefaultPosition, const wxSize& size
78         = wxDefaultSize, long style = wxDEFAULT_DIALOG_STYLE);
79 private:
80     circuit* c;
81     wxListBox *lbox;
82     void OnOK(wxCommandEvent& event);
83     DECLARE_EVENT_TABLE()
84 };
85
86 #endif /* gui_h */

```

Listing 13: gui.cc

```

1 #include "gui.h"
2 #include "wx_icon.xpm"
3 #include <iostream>
4 #include <vector>
5 #include <algorithm>
6 #include "scanner.h"
7 #include "parser.h"
8 #include "gui-devices.h"
9 #include "gui-id.h"
10
11 using namespace std;
12
13 // MyFrame
14 ///////////////////////////////////////////////////////////////////
15
16 BEGIN_EVENT_TABLE(MyFrame, wxFrame)
17     EVT_MENU(wxID_EXIT, MyFrame::OnExit)
18     EVT_MENU(wxID_ABOUT, MyFrame::OnAbout)
19     EVT_MENU(MENU_CLEAR_CIRCUIT, MyFrame::OnMenuClearCircuit)
20     EVT_MENU(wxID_OPEN, MyFrame::OnOpenFile)
21     EVT_BUTTON(SIMCTRL_BUTTON_RUN_ID, MyFrame::OnButtonRun)
22     EVT_BUTTON(SIMCTRL_BUTTON_CONT_ID, MyFrame::OnButtonContinue)
23     EVT_BUTTON(MONITORS_ADD_BUTTON_ID, MyFrame::OnButtonAddMon)
24     EVT_BUTTON(MONITORS_DEL_BUTTON_ID, MyFrame::OnButtonDelMon)
25     EVT_BUTTON(DEVICES_EDIT_BUTTON_ID, MyFrame::OnButtonEditDevs)
26 END_EVENT_TABLE()
27
28 MyFrame::MyFrame(wxWindow *parent, const wxString& title, const
29     wxPoint& pos, const wxSize& size,
30     names *names_mod, devices *devices_mod, monitor *
31     monitor_mod, network *net_mod, long style):
32     wxFrame(parent, wxID_ANY, title, pos, size, style)
33     // Constructor - initialises pointers to names, devices and monitor
34     // classes, lays out widgets
35     // using sizers
36 {
37     SetIcon(wxIcon(wx_icon));
38
39     if (names_mod == NULL || devices_mod == NULL || monitor_mod == NULL
40         || net_mod == NULL) {
41         cout << "Cannot operate GUI without names, devices, network and
42             monitor classes" << endl;
43         exit(1);
44     }
45
46     c = new circuit(names_mod, devices_mod, monitor_mod, net_mod);
47
48     // Menu items
49     wxMenu *fileMenu = new wxMenu;

```



```

45 fileMenu->Append(wxID_OPEN);
46 fileMenu->Append(MENU_CLEAR_CIRCUIT, _("Clear circuit"));
47 fileMenu->AppendSeparator();
48 fileMenu->Append(wxID_ABOUT, _("&About"));
49 fileMenu->Append(wxID_EXIT, _("&Quit"));
50 wxMenuBar *menuBar = new wxMenuBar;
51 menuBar->Append(fileMenu, _("&File"));
52 SetMenuBar(menuBar);
53
54 // Everything is contained in a wxPanel for improved appearance in
55 // wine/windows, as
56 // described in http://wiki.wxwidgets.org/WxFAQ
57 wxPanel* mainPanel = new wxPanel(this, wxID_ANY, wxDefaultPosition,
58 wxDefaultSize, wxTAB_TRAVERSAL);
59 wxBoxSizer *topSizer = new wxBoxSizer(wxHORIZONTAL);
60 wxBoxSizer *leftSizer = new wxBoxSizer(wxVERTICAL);
61
62 // Canvas for drawing monitor traces
63 canvas = new MyGLCanvas(c, mainPanel, wxID_ANY, wxDefaultPosition,
64 wxDefaultSize, wxBORDER_SUNKEN);
65 leftSizer->Add(canvas, 3, wxEXPAND | wxALL, 10);
66
67 // Create the log textbox, mainly for displaying error messages
68 // from the parser, captures everything sent to cout
69 // wxTE_DONTWRAP means that a horizontal scrollbar will be used
70 // instead of wrapping, so that the position of an error can be
71 // indicated correctly
72 outputTextCtrl = new wxTextCtrl(mainPanel, OUTPUT_TEXTCTRL_ID, wxT(
73 ""), wxDefaultPosition, wxDefaultSize, wxTE_MULTILINE |
74 wxTE_READONLY | wxTE_DONTWRAP);
75
76 // Set log textbox to a monospace font, so that the position of an
77 // error can be indicated correctly
78 wxTextAttr outputTextAttr = outputTextCtrl->GetDefaultStyle();
79 outputTextAttr.SetFont(wxFont(10, wxFAMILY_MODERN,
80 wxFONTSTYLE_NORMAL, wxFONTWEIGHT_NORMAL));
81 outputTextCtrl->SetDefaultStyle(outputTextAttr);
82
83 // Redirect all text sent to cout to the log textbox
84 outputTextRedirect = new wxStreamToTextRedirector(outputTextCtrl);
85 leftSizer->Add(outputTextCtrl, 1, wxEXPAND | wxALL, 10);
86 topSizer->Add(leftSizer, 4, wxEXPAND | wxALL, 10);
87
88 // Simulation controls box
89 wxBoxSizer *sidesizer = new wxBoxSizer(wxVERTICAL);
90 simctrls_container = new wxPanel(mainPanel, wxID_ANY);
91 wxStaticBoxSizer *simctrls_sizer = new wxStaticBoxSizer(wxVERTICAL,
92 simctrls_container, _("Simulation"));
93 wxBoxSizer *simctrls_cycles_sizer = new wxBoxSizer(wxHORIZONTAL);
94 wxBoxSizer *simctrls_button_sizer = new wxBoxSizer(wxHORIZONTAL);
95
96 // Run and continue simulation buttons
97 simctrl_run = new wxButton(simctrls_container,
98 SIMCTRL_BUTTON_RUN_ID, _("Run"));
99 simctrl_continue = new wxButton(simctrls_container,
100 SIMCTRL_BUTTON_CONT_ID, _("Continue"));
101 simctrls_button_sizer->Add(simctrl_run, 0, wxALL, 10);
102 simctrls_button_sizer->Add(simctrl_continue, 0, wxALL, 10);
103 // Simulation cycle number spinner
104 simctrls_cycles_sizer->Add(new wxStaticText(simctrls_container,
105 wxID_ANY, _("Cycles")), 0, wxALL | wxALIGN_CENTER_VERTICAL, 10);
106 spin = new wxSpinCtrl(simctrls_container, MY_SPINCTRL_ID, wxString(
107 wxT("42")));
108 spin->SetRange(1,1000000);
109 simctrls_cycles_sizer->Add(spin, 0, wxALL, 10);
110
111 simctrls_sizer->Add(simctrls_cycles_sizer);
112 simctrls_sizer->Add(simctrls_button_sizer);
113 simctrls_container->SetSizerAndFit(simctrls_sizer);
114
115 // Buttons to open add/remove monitor dialogs
116 wxStaticBoxSizer *edit_sizer = new wxStaticBoxSizer(wxVERTICAL,
117 mainPanel, _("Edit circuit"));
118 monitors_add_btn = new wxButton(mainPanel, MONITORS_ADD_BUTTON_ID,
119 _("Add monitors"));
120 monitors_rem_btn = new wxButton(mainPanel, MONITORS_DEL_BUTTON_ID,
121 _("Remove monitors"));
122 edit_sizer->Add(monitors_add_btn, 0, (wxALL & ~wxBOTTOM) | wxEXPAND
123 , 10);
124 edit_sizer->Add(monitors_rem_btn, 0, wxLEFT | wxRIGHT | wxEXPAND,
125 10);
126 edit_sizer->Add(new wxButton(mainPanel, DEVICES_EDIT_BUTTON_ID, _("
127 Edit devices")), 0, (wxALL & ~wxTOP) | wxEXPAND, 10);
128
129 // wxCheckBox that allows switch states to be changed
130 wxStaticBoxSizer *switches_sizer = new wxStaticBoxSizer(wxVERTICAL,
131 mainPanel, _("Switches"));
132 switchesCtrl = new SwitchesCheckBox(c, mainPanel,
133 SWITCHES_CTRL_ID, wxDefaultPosition, wxDefaultSize,
134 wxLB_NEEDED_SB);
135 switches_sizer->Add(switchesCtrl, 1, wxEXPAND | wxALL, 10);
136
137 sidesizer->Add(simctrls_container, 0, wxALL, 10);
138 sidesizer->Add(edit_sizer, 0, wxEXPAND | wxALL, 10);
139 sidesizer->Add(switches_sizer, 1, wxEXPAND | wxALL, 10);
140 topSizer->Add(sidesizer, 0, wxEXPAND | wxALIGN_CENTER);
141
142 SetSizeHints(400, 400);
143 mainPanel->SetSizer(topSizer);
144
145 c->circuitChanged.Attach(this, &MyFrame::UpdateControlStates);

```

```

123 c->monitorsChanged.Attach(this, &MyFrame::UpdateControlStates);
124 c->monitorSamplesChanged.Attach(this, &MyFrame::UpdateControlStates
    );
125 UpdateControlStates();
126 }
127
128 MyFrame::~MyFrame()
129 {
130     c->circuitChanged.Detach(this);
131     c->monitorsChanged.Detach(this);
132     c->monitorSamplesChanged.Detach(this);
133     delete outputTextRedirect;
134 }
135
136 void MyFrame::OnExit(wxCommandEvent &event)
137 // Callback for the exit menu item
138 {
139     Close(true);
140 }
141
142 void MyFrame::OnAbout(wxCommandEvent &event)
143 // Callback for the about menu item
144 {
145     wxMessageDialog about(this, _("Logic simulator\nIIA GF2 Team 8\
n2013"), _("About Logsim"), wxICON_INFORMATION | wxOK);
146     about.ShowModal();
147 }
148
149 void MyFrame::OnMenuClearCircuit(wxCommandEvent &event)
150 {
151     c->Clear();
152 }
153
154 void MyFrame::OnOpenFile(wxCommandEvent &event)
155 // Callback for the File -> Open menu item
156 {
157     wxFileDialog openFileDialog(this, _("Open logic circuit"), wxT("../
examples/"), wxT(""), _("Logic circuit files (*.gf2)|*.gf2|All
files (*.*)|*.*"), wxFD_OPEN | wxFD_FILE_MUST_EXIST |
wxFD_CHANGE_DIR);
158     if (openFileDialog.ShowModal() == wxID_CANCEL)
159         return; // cancelled, don't open a file
160     loadFile(openFileDialog.GetPath().mb_str());
161 }
162
163 bool MyFrame::loadFile(const char * filename)
164 // load a file (can be called by menu File->Open or for the command
    line argument)
165 {
166     bool result; //True if file is opened correctly
167     // Clear log window

```

```

168     outputTextCtrl->ChangeValue(wxT(""));
169     cout << "Loading file " << filename << endl;
170
171     c->Clear();
172
173     scanner *smz = new scanner(c->nmz(), filename, result);
174     error *erz = new error(smz);
175     parser *pmz = new parser(c->netz(), c->dmz(), c->mmz(), smz, erz);
176     if (result) result = pmz->readin();
177
178     if (result)
179     {
180         c->netz()->checknetwork(result);
181     }
182
183     if (!result)
184     {
185         cout << "Failed to load file" << endl;
186         c->Clear();
187
188         // scroll to the start of the output so that the first error
            message (which may have caused any subsequent error messages) is
            visible
189         outputTextCtrl->ShowPosition(0);
190     }
191
192     c->circuitChanged.Trigger();
193     c->monitorsChanged.Trigger();
194     c->monitorSamplesChanged.Trigger();
195
196     if (!result)
197     {
198         canvas->SetErrorMessage(_("Failed to load file"));
199     }
200
201     delete pmz;
202     delete erz;
203     delete smz;
204
205     return result;
206 }
207
208 void MyFrame::UpdateControlStates()
209 {
210     // Update enabled/disabled state of controls
211     if (c->netz()->devicelist()==NULL)
212     {
213         // If there are no devices, disable simulation controls and add/
            remove monitor buttons
214         simctrls_container->Disable();
215         monitors_add_btn->Disable();

```

```

216     monitors_rem_btn->Disable();
217 }
218 else
219 {
220     // The circuit contains some devices, so enable the simulation
221     // controls if all inputs are connected
222     // It is assumed that unconnected inputs are listed by whichever
223     // bit of code has allowed them to exist in the circuit, this
224     // function just updates control stated
225     bool ok;
226     c->netz()->checknetwork(ok, true);
227     if (ok)
228     {
229         simctrls_container->Enable();
230     }
231     else
232     {
233         simctrls_container->Disable();
234         // Only enable the add monitors button if some unmonitored
235         // outputs exist
236         if (c->GetUnmonitoredOutputs())
237             monitors_add_btn->Enable();
238         else
239             monitors_add_btn->Disable();
240         // Only enable the remove monitors button if some monitors exist
241         if (c->mmz()->moncount()>0)
242             monitors_rem_btn->Enable();
243         else
244             monitors_rem_btn->Disable();
245         // Disable the continue button if the run button has not been
246         // used first (so GetTotalCycles returns 0) or there are some new
247         // monitors (so some monitors do not contain any samples)
248         bool someEmpty = (c->GetTotalCycles()==0);
249         for (int i=0; i<c->mmz()->moncount(); i++)
250         {
251             if (c->mmz()->getsamplecount(i)==0)
252                 someEmpty = true;
253         }
254         if (someEmpty)
255             simctrl_continue->Disable();
256         else
257             simctrl_continue->Enable();
258     }
259 }
260 void MyFrame::OnButtonRun(wxCommandEvent &event)
261 // Callback for the run simulation button
262 {
263     c->Simulate(spin->GetValue(),true);
264     simctrl_continue->Enable();
265 }
266 void MyFrame::OnButtonContinue(wxCommandEvent &event)

```

```

261 // Callback for the continue simulation button
262 {
263     c->Simulate(spin->GetValue(),false);
264 }
265 void MyFrame::OnButtonAddMon(wxCommandEvent& event)// "Add monitors"
266     button clicked
267 {
268     int oldMonCount = c->mmz()->moncount();
269     AddMonitorsDialog *dlg = new AddMonitorsDialog(c, this, _("Add
270     monitors"), wxDefaultPosition, wxDefaultSize,
271     wxDEFAULT_DIALOG_STYLE | wxRESIZE_BORDER);
272     if (dlg->ShowModal()==wxID_OK && oldMonCount!=c->mmz()->moncount())
273     {
274         if (c->GetTotalCycles())
275         {
276             cout << wxString(wxPLURAL("Monitor added, run simulation again
277             to see updated signals", "Monitors added, run simulation again to
278             see updated signals", c->mmz()->moncount()-oldMonCount)).mb_str
279             () << endl;
280         }
281     }
282     dlg->Destroy();
283 }
284 void MyFrame::OnButtonDelMon(wxCommandEvent& event)// "Remove
285     monitors" button clicked
286 {
287     DelMonitorsDialog *dlg = new DelMonitorsDialog(c, this, _("Remove
288     monitors"), wxDefaultPosition, wxDefaultSize,
289     wxDEFAULT_DIALOG_STYLE | wxRESIZE_BORDER);
290     dlg->ShowModal();
291     dlg->Destroy();
292 }
293 void MyFrame::OnButtonEditDevs(wxCommandEvent& event)// "Edit devices
294     " button clicked
295 {
296     DevicesDialog *dlg = new DevicesDialog(c, this, wxID_ANY, _("Edit
297     devices"));
298     dlg->ShowModal();
299     dlg->Destroy();
300     bool ok;
301     c->netz()->checknetwork(ok);
302     if (!ok) canvas->SetErrorMessage(_("Circuit contains unconnected
303     inputs"));
304     else canvas->ClearErrorMessage();
305     UpdateControlStates();
306 }

```

```

300 AddMonitorsDialog::AddMonitorsDialog(circuit* circ, wxWindow* parent,
301     const wxString& title, const wxPoint& pos, const wxSize& size,
302     long style):
303     wxDialog(parent, wxID_ANY, title, pos, size, style)
304 {
305     SetIcon(wxIcon(wx_icon));
306
307     c = circ;
308     oldMonCount = c->mmz()->moncount();
309
310     // Make a list of unmonitored outputs
311     c->GetUnmonitoredOutputs(&availableOutputs);
312     sort(availableOutputs.begin(), availableOutputs.end(),
313         CircuitElementInfo_namestrcmp);
314     wxArrayString displayedOutputs;
315     CircuitElementInfoVector_to_wxArrayString(availableOutputs,
316         displayedOutputs);
317
318     wxBoxSizer* topsizer = new wxBoxSizer(wxVERTICAL);
319     topsizer->Add(new wxStaticText(this, wxID_ANY, _("Select output(s)
320         to monitor:")), 0, wxLEFT | wxRIGHT | wxTOP, 10);
321     lbox = new wxListBox(this, wxID_ANY, wxDefaultPosition,
322         wxDefaultSize, displayedOutputs, wxLB_EXTENDED | wxLB_NEEDED_SB);
323     topsizer->Add(lbox, 1, wxALL | wxEXPAND, 10);
324     topsizer->Add(CreateButtonSizer(wxOK | wxCANCEL), 0, wxALL |
325         wxEXPAND, 10);
326     SetSizerAndFit(topsizer);
327 }
328
329 void AddMonitorsDialog::OnOK(wxCommandEvent& event)
330 {
331     wxArrayInt selections;
332     lbox->GetSelections(selections);
333     int num = selections.GetCount();
334     bool ok;
335     for (int i=0; i<num; i++)
336     {
337         c->mmz()->makemonitor(availableOutputs[selections[i]].d->id,
338             availableOutputs[selections[i]].o->id, ok);
339     }
340     if (c->mmz()->moncount() != oldMonCount)
341     {
342         c->monitorsChanged.Trigger();
343     }
344     EndModal(wxID_OK);
345 }
346
347 BEGIN_EVENT_TABLE(AddMonitorsDialog, wxDialog)
348     EVT_BUTTON(wxID_OK, AddMonitorsDialog::OnOK)
349 END_EVENT_TABLE()

```

```

343 DelMonitorsDialog::DelMonitorsDialog(circuit* circ, wxWindow* parent,
344     const wxString& title, const wxPoint& pos, const wxSize& size,
345     long style):
346     wxDialog(parent, wxID_ANY, title, pos, size, style)
347 {
348     SetIcon(wxIcon(wx_icon));
349     c = circ;
350
351     // Copy monitor names into a wxArrayString to pass to the listbox
352     int monCount = c->mmz()->moncount();
353     wxArrayString monitorList;
354     monitorList.Alloc(monCount);
355     for (int i=0; i<monCount; i++)
356     {
357         monitorList.Add(wxString(c->mmz()->getsignalstring(i).c_str(),
358             wxConvUTF8));
359     }
360
361     // Create controls
362     wxBoxSizer* topsizer = new wxBoxSizer(wxVERTICAL);
363     topsizer->Add(new wxStaticText(this, wxID_ANY, _("Select monitors
364         to remove:")), 0, wxLEFT | wxRIGHT | wxTOP, 10);
365     lbox = new wxListBox(this, wxID_ANY, wxDefaultPosition,
366         wxDefaultSize, monitorList, wxLB_EXTENDED | wxLB_NEEDED_SB);
367     topsizer->Add(lbox, 1, wxALL | wxEXPAND, 10);
368     topsizer->Add(CreateButtonSizer(wxOK | wxCANCEL), 0, wxALL |
369         wxEXPAND, 10);
370     SetSizerAndFit(topsizer);
371 }
372
373 void DelMonitorsDialog::OnOK(wxCommandEvent& event)
374 {
375     wxArrayInt selections;
376     lbox->GetSelections(selections);
377     bool ok;
378     monitor* mmz = c->mmz();
379     name dev, outp;
380     // Remove selected monitors
381     // Note loop direction - remmonitor deletes an entry and shifts
382     // monitors into the space, changing the IDs of later monitors.
383     // Therefore remove monitors with a higher ID (towards the end of
384     // the listbox) first.
385     for (int i=selections.GetCount()-1; i>=0; i--)
386     {
387         mmz->getmonname(selections[i], dev, outp);
388         mmz->remmonitor(dev, outp, ok);
389     }
390     if (selections.GetCount())
391         c->monitorsChanged.Trigger();

```

```

385 EndModal(wxID_OK);
386 }
387 }
388
389 BEGIN_EVENT_TABLE(DelMonitorsDialog, wxDialog)
390     EVT_BUTTON(wxID_OK, DelMonitorsDialog::OnOK)
391 END_EVENT_TABLE()

```

### 3.2.4 Main GUI: monitor traces canvas

Listing 14: gui-canvas.h

```

1  #ifndef gui_canvas_h
2  #define gui_canvas_h
3
4  #include "circuit.h"
5  #include "gui-misc.h"
6  #include <wx/wx.h>
7  #include <wx/glcanvas.h>
8
9  int GetGlutTextWidth(wxString txt, void *font=NULL);
10 void DrawGlutText(int x, int y, wxString txt, void *font=NULL);
11
12 class MyGLCanvas;
13
14 void wxRect_GlVertex(const wxRect& r);
15
16 class GLCanvasMonitorTrace
17 {
18 public:
19     GLCanvasMonitorTrace();
20     GLCanvasMonitorTrace(int newMonId, circuit* c);
21     // Get or set the monitor id (the "n" in "n'th monitor" in monitor
22     // class calls, also determines position)
23     int GetMonitorId();
24     void SetMonitorId(int newMonId);
25     // Notify of a change in the number of displayed cycles
26     void OnMonitorSamplesChanged(int totalCycles_new, int
27     continuedCycles_new);
28     // Draw the signal trace, visible coordinates are used so that time
29     // is not wasted in drawing areas hidden due to scrolling.
30     void Draw(MyGLCanvas *canvas, const wxRect& visibleRegion);
31     void DrawName(MyGLCanvas *canvas, const wxRect& visibleRegion);
32     // Get the width in pixels of the name, used by MyGLCanvas.Render()
33     // to determine how much space to leave between the traces and the
34     // edge of the canvas
35     int GetNameWidth();
36     // Set the geometry and positioning of the monitor trace.

```

```

32 // xOffset - the x position of the graph origin
33 // yCentre - the y position of the centre of the drawn signal (
34 // halfway between high and low signal levels)
35 // xScale - the x-axis scale (the number of pixels per cycle).
36 // height - the height of the signal trace line itself,
37 // padding - the distance between the top of the signal trace and
38 // the edge of the graph background
39 // spacing - the vertical distance between centre lines of
40 // consecutive monitor traces.
41 // xBgName - the distance between the edge of the screen and the
42 // faint signal name drawn on top of the traces when the text to the
43 // left is invisible.
44 void SetGeometry(int xOffset_new, int yCentre_new, double
45 xScale_new, int sigHeight_new, int padding_new, int spacing_new,
46 int xBgName_new, int axisLabelInterval_new);
47 private:
48     int monId;
49     circuit* c;
50     wxString monName; // cached monitor name, to avoid constructing it
51     // again every time the signal is drawn
52     int monNameWidth; // width in pixels of monitor name
53     bool geometrySet; // whether SetGeometry() has been called
54
55     // variables controlling position and size of the trace, see
56     // SetGeometry() for details
57     int xOffset, yCentre, sigHeight, padding, spacing, xBgName;
58     int axisLabelInterval; // cycles between numbers on x axis
59     double xScale;
60
61     void UpdateName(); // Update monName and monNameWidth
62     int continuedCycles; // how many simulation cycles were completed
63     // last time the run or continue button was used
64     int totalCycles; // how many simulation cycles have been completed
65     // in total
66 };
67
68 class MyGLCanvas: public wxGLCanvas, public wxScrollHelperNative
69 {
70 public:
71     MyGLCanvas(circuit* circ, wxWindow *parent, wxWindowID id =
72     wxID_ANY, const wxPoint& pos = wxDefaultPosition, const wxSize&
73     size = wxDefaultSize, long style = 0, const wxString& name = wxT(
74     "MyGLCanvas"));
75     ~MyGLCanvas();
76     void Render(); // function to draw canvas contents
77     void OnMonitorSamplesChanged();
78     void OnMonitorsChanged();
79     void UpdateMinCanvasSize();
80     void SetErrorMessage(wxString txt);
81     void ClearErrorMessage();

```



```

69 private:
70     bool init;                // has the GL context been
71     initialised?
72     circuit* c;
73     int maxMonNameWidth;
74     vector<GLCanvasMonitorTrace> mons;
75     void InitGL();            // function to initialise GL
76     context
77     void OnSize(wxSizeEvent& event); // callback for when canvas is
78     resized
79     void OnPaint(wxPaintEvent& event); // callback for when canvas is
80     exposed
81     void OnMouse(wxMouseEvent& event); // callback for mouse events
82     inside canvas
83     void DrawInfoTextCentre(wxString txt, bool isError = false);
84     void Redraw();
85     wxString errorMessage;
86     int scrollX, scrollY;
87     int minXScale, maxXScale;
88 public:
89     virtual void ScrollWindow(int dx, int dy, const wxRect* rect = (
90     wxRect *)NULL);
91
92     WX_FORWARD_TO_SCROLL_HELPER() // copied from wxScrolledWindow
93
94 private:
95     DECLARE_EVENT_TABLE()
96 };
97
98 #endif /* gui_canvas_h */

```

Listing 15: gui-canvas.cc

```

1  #include "gui-canvas.h"
2  #include <GL/glut.h>
3  #include <cmath>
4
5  // Get the width in pixels of some text when drawn using a particular
6  font (font defaults to GLUT_BITMAP_HELVETICA_12)
7  int GetGlutTextWidth(wxString txt, void *font)
8  {
9      int width = 0;
10     if (!font) font = GLUT_BITMAP_HELVETICA_12;
11     for (int i = 0; i < txt.Len(); i++)
12         width += glutBitmapWidth(font, txt[i]);
13     return width;
14 }

```

```

15 // Draws some text at the given position (font defaults to
16 GLUT_BITMAP_HELVETICA_12)
17 void DrawGlutText(int x, int y, wxString txt, void *font)
18 {
19     if (!font) font = GLUT_BITMAP_HELVETICA_12;
20     int xMove = 0, yMove = 0;
21     /* If x or y are outside the viewport, call glRasterPos2f with a
22     position inside
23     * the viewport, then move it using glBitmap (as described in
24     glRasterPos man page).
25     * Otherwise the text doesn't get rendered.
26     * Assuming viewport starts at (0,0) and only implemented for
27     positions to the left of
28     * or above (0,0) for now. */
29     if (x<0) xMove = x;
30     if (y<0) yMove = y;
31     glRasterPos2f(x-xMove, y-yMove);
32     if (xMove || yMove) glBitmap(0, 0, 0, 0, xMove, yMove, NULL);
33     for (int i = 0; i < txt.Len(); i++)
34         glutBitmapCharacter(font, txt[i]);
35 }
36
37 // Call glVertex for the positions of the corners of a wxRect
38 // This makes drawing boxes a bit simpler
39 void wxRect_GlVertex(const wxRect& r)
40 {
41     glVertex2i(r.x, r.y);
42     glVertex2i(r.x+r.width, r.y);
43     glVertex2i(r.x+r.width, r.y+r.height);
44     glVertex2i(r.x, r.y+r.height);
45 }
46
47 // GLCanvasMonitorTrace - class to handle drawing of one monitor
48 trace
49 GLCanvasMonitorTrace::GLCanvasMonitorTrace() :
50     monId(-1), c(NULL), monName(wxT("")), monNameWidth(0), geometrySet(
51     false)
52 {}
53
54 GLCanvasMonitorTrace::GLCanvasMonitorTrace(int newMonId, circuit*
55     circ) :
56     geometrySet(false)
57 {
58     c = circ;
59     SetMonitorId(newMonId);
60 }
61
62 int GLCanvasMonitorTrace::GetMonitorId()
63 {
64     return monId;
65 }

```

```

59 void GLCanvasMonitorTrace::SetMonitorId(int newMonId)
60 {
61     monId = newMonId;
62     UpdateName();
63 }
64
65 void GLCanvasMonitorTrace::OnMonitorSamplesChanged(int
66     totalCycles_new, int continuedCycles_new)
67 {
68     totalCycles = totalCycles_new;
69     continuedCycles = continuedCycles_new;
70 }
71
72 void GLCanvasMonitorTrace::UpdateName()
73 {
74     if (!c)
75     {
76         monName = wxT("");
77         monNameWidth = 0;
78         return;
79     }
80     monName = wxString(c->mmz()->getsignalstring(monId).c_str(),
81         wxConvUTF8);
82     monNameWidth = GetGlutTextWidth(monName, GLUT_BITMAP_HELVETICA_12);
83 }
84
85 int GLCanvasMonitorTrace::GetNameWidth()
86 {
87     return monNameWidth;
88 }
89
90 void GLCanvasMonitorTrace::SetGeometry(int xOffset_new, int
91     yCentre_new, double xScale_new, int sigHeight_new, int
92     padding_new, int spacing_new, int xBgName_new, int
93     axisLabelInterval_new)
94 {
95     geometrySet = true;
96     xOffset = xOffset_new;
97     yCentre = yCentre_new;
98     xScale = xScale_new;
99     sigHeight = int((sigHeight_new/2)*2); // make sure this is even, so
100     that the trace is symmetrical about the horizontal centre line
101     padding = padding_new;
102     spacing = spacing_new;
103     xBgName = xBgName_new;
104     axisLabelInterval = axisLabelInterval_new;
105 }
106
107 void GLCanvasMonitorTrace::Draw(MyGLCanvas *canvas, const wxRect&
108     visibleRegion)

```

```

103 {
104     if (!c || !canvas || monId<0 || monId>=c->mmz()->moncount() || !
105         geometrySet) return;
106
107     // If this monitor was added after the simulation was run, it might
108     // have no data even though c->GetTotalCycles() is greater than
109     // zero
110     int sampleCount = c->mmz()->getsamplecount(monId);
111     if (!sampleCount)
112     {
113         glColor4f(0.8, 0.0, 0.0, 1.0);
114         if (xOffset < visibleRegion.x+xBgName)
115         {
116             DrawGlutText(xBgName, yCentre-5, _("No data"),
117                 GLUT_BITMAP_HELVETICA_12);
118         }
119         else
120         {
121             DrawGlutText(xOffset+10, yCentre-5, _("No data"),
122                 GLUT_BITMAP_HELVETICA_12);
123         }
124     }
125     if (totalCycles > sampleCount)
126         totalCycles = sampleCount;
127     if (continuedCycles > sampleCount)
128         continuedCycles = sampleCount;
129
130     wxRect backgroundRegion(xOffset, yCentre-sigHeight/2-padding, ceil(
131         xScale*totalCycles), sigHeight+padding*2);
132     wxRect traceRegion = wxRect(xOffset, yCentre-sigHeight/2-padding
133         -11, ceil(xScale*totalCycles), sigHeight+padding*2+11); //
134     includes cycle numbers on the axis
135     if (traceRegion.Intersect(visibleRegion).IsEmpty()) return;
136     wxRect clippedbg = backgroundRegion;
137     clippedbg.Intersect(visibleRegion);
138
139     // background colour
140     glBegin(GL_QUADS);
141     glColor4f(0.0, 0.5, 0.0, 0.08);
142     wxRect_GlVertex(clippedbg);
143     glEnd();
144
145     // border
146     glColor4f(0.0, 0.7, 0.0, 0.4);
147     glBegin(GL_LINE_LOOP);
148     wxRect_GlVertex(backgroundRegion);
149     glEnd();
150
151     // actual signal trace
152     if (xScale>5) glLineWidth(2);
153     glBegin(GL_LINE_STRIP);

```

```

146 glColor4f(0.0, 0.8, 0.0, 1.0);
147 int y1, y2, i;
148 asignal s;
149 int firstCycle = 0;
150 int cycleLimit = totalCycles;
151 if (xOffset < visibleRegion.x)
152     firstCycle = int((visibleRegion.x-xOffset)/xScale);
153 if (xOffset + totalCycles*xScale > visibleRegion.x+visibleRegion.
154     width)
155     cycleLimit = int((visibleRegion.x+visibleRegion.width - xOffset)/
156     xScale) + 1;
157 if (cycleLimit>totalCycles)
158     cycleLimit = totalCycles;
159 int prevY = yCentre;
160 for (i=firstCycle; i<cycleLimit; i++)
161 {
162     if (c->mmz()->getsignaltrace(monId, i, s))
163     {
164         if (s==low || s==rising)
165             y1 = yCentre-sigHeight/2;
166         else
167             y1 = yCentre+sigHeight/2;
168         if (s==low || s==falling)
169             y2 = yCentre-sigHeight/2;
170         else
171             y2 = yCentre+sigHeight/2;
172         if (y1!=prevY) glVertex2i(xOffset+xScale*i, y1);
173         glVertex2i(xOffset+xScale*(i+1), y2);
174         prevY = y2;
175     }
176 }
177 glEnd();
178 glLineWidth(1);
179 // Draw cycle numbers on axis and dashed vertical lines for them
180 firstCycle = (int(firstCycle/axisLabelInterval)-1) *
181     axisLabelInterval;
182 if (firstCycle<0)
183     firstCycle = 0;
184 cycleLimit = (int(cycleLimit/axisLabelInterval)+1) *
185     axisLabelInterval;
186 if (cycleLimit>totalCycles)
187     cycleLimit = totalCycles;
188 glLineStipple(2, 0xAAAA);
189 glEnable(GL_LINE_STIPPLE);
190 for (i=firstCycle; i<=cycleLimit; i+=axisLabelInterval)
191 {
192     if (i!=0 && i!=totalCycles)
193     {
194         glBegin(GL_LINE_STRIP);
195         glColor4f(0.0, 0.0, 0.0, 0.2);

```

```

196         glVertex2i(xOffset+xScale*i, yCentre-sigHeight/2-padding);
197         glVertex2i(xOffset+xScale*i, yCentre+sigHeight/2+padding);
198         glEnd();
199     }
200     glColor4f(0.0, 0.0, 0.0, 1.0);
201     wxString labelText;
202     labelText.Printf(wxT("%d"), i);
203     int labelWidth = GetGlutTextWidth(labelText,
204     GLUT_BITMAP_HELVETICA_10);
205     DrawGlutText(xOffset+xScale*i - labelWidth/2, yCentre-sigHeight
206     /2-padding-11, labelText, GLUT_BITMAP_HELVETICA_10);
207 }
208 glDisable(GL_LINE_STIPPLE);
209 }
210 void GLCanvasMonitorTrace::DrawName(MyGLCanvas *canvas, const wxRect&
211     visibleRegion)
212 {
213     if (!geometrySet) return;
214     if (xOffset < visibleRegion.x+xBgName && c->mmz()->getsamplecount(
215     monId))
216     {
217         glColor4f(0.0, 0.0, 0.0, 0.4);
218         DrawGlutText(xBgName, yCentre-5, monName,
219         GLUT_BITMAP_HELVETICA_12);
220     }
221     else
222     {
223         glColor4f(0.0, 0.0, 0.0, 1.0);
224         DrawGlutText(xOffset-monNameWidth-5, yCentre-5, monName,
225         GLUT_BITMAP_HELVETICA_12);
226     }
227 }
228 // MyGLCanvas
229 BEGIN_EVENT_TABLE(MyGLCanvas, wxGLCanvas)
230     EVT_SIZE(MyGLCanvas::OnSize)
231     EVT_PAINT(MyGLCanvas::OnPaint)
232     EVT_MOUSE_EVENTS(MyGLCanvas::OnMouse)
233 END_EVENT_TABLE()
234 int wxglcanvas_attr_list[5] = {WX_GL_RGBA, WX_GL_DOUBLEBUFFER,
235     WX_GL_DEPTH_SIZE, 16, 0};
236 MyGLCanvas::MyGLCanvas(circuit* circ, wxWindow *parent, wxWindowID id
237     ,
238     const wxPoint& pos, const wxSize& size, long
239     style, const wxString& name):

```



```

233 wxGLCanvas(parent, id, pos, size, style, name,
      wxglcanvas_attrib_list),
234 wxScrollHelperNative(this), scrollX(0), scrollY(0)
235 // Constructor - initialises private variables
236 {
237     init = false;
238     c = circ;
239     if (c)
240     {
241         c->monitorsChanged.Attach(this, &MyGLCanvas::OnMonitorsChanged);
242         c->monitorSamplesChanged.Attach(this, &MyGLCanvas::
            OnMonitorSamplesChanged);
243         c->monitorSamplesChanged.Attach(this, &MyGLCanvas::
            ClearErrorMessage);
244         OnMonitorsChanged();
245     }
246     SetScrollRate(10,10);
247     minXScale = 2;
248     maxXScale = 50;
249 }
250
251 MyGLCanvas::~MyGLCanvas()
252 {
253     if (c)
254     {
255         c->monitorsChanged.Detach(this);
256         c->monitorSamplesChanged.Detach(this);
257     }
258 }
259
260 void MyGLCanvas::Redraw()
261 {
262     // Redraw the contents of the canvas
263     // This might look bizarre, but it seems to work everywhere tested
264     // (linux, and a cross compiled executable running in wine)
265     // Removing any of these calls mean the canvas is sometimes not
266     // redrawn when running in wine
267     Refresh();
268     Update();
269     Refresh();
270 }
271
272 void MyGLCanvas::ScrollWindow(int dx, int dy, const wxRect *rect)
273 {
274     // override ScrollWindow(), since MyGLCanvas does its own scrolling
275     // by offsetting all drawn points and clipping
276     scrollX += dx;
277     scrollY += dy;
278     Redraw();
279 }

```

```

278 void MyGLCanvas::UpdateMinCanvasSize()
279 {
280     int totalCycles = c->GetTotalCycles();
281     if (!mons.size())
282         totalCycles = 0; // If all monitors have been deleted, don't
283         // reserve horizontal space
284     // Make sure all the traces fit in the canvas
285     int xOffset = maxMonNameWidth+5;
286     int maxXTextWidth = ceil(log10(totalCycles))*8; // estimate of max x
287         // axis scale text width
288     SetVirtualSize(minXScale*totalCycles+15+xOffset+maxXTextWidth/2, 50*
289         mons.size()+10);
290 }
291
292 // Notify of a change to the number of displayed cycles
293 void MyGLCanvas::OnMonitorSamplesChanged()
294 {
295     int totalCycles = c->GetTotalCycles();
296     int continuedCycles = c->GetContinuedCycles();
297     for (int i=0; i<mons.size(); i++)
298     {
299         mons[i].OnMonitorSamplesChanged(totalCycles, continuedCycles);
300     }
301     UpdateMinCanvasSize();
302     // Scroll to the most recently simulated cycles
303     Scroll(GetVirtualSize().GetWidth()-GetClientSize().GetWidth(), -1);
304     Redraw();
305 }
306
307 // Notify of a change to the active monitors
308 void MyGLCanvas::OnMonitorsChanged()
309 {
310     int monCount = c->mmz()->moncount();
311     mons.resize(monCount);
312     maxMonNameWidth = 0;
313     int totalCycles = c->GetTotalCycles();
314     int continuedCycles = c->GetContinuedCycles();
315     for (int i=0; i<monCount; i++)
316     {
317         mons[i] = GLCanvasMonitorTrace(i, c);
318         mons[i].OnMonitorSamplesChanged(totalCycles, continuedCycles);
319         if (mons[i].GetNameWidth()>maxMonNameWidth)
320             maxMonNameWidth = mons[i].GetNameWidth();
321     }
322     UpdateMinCanvasSize();
323     Redraw();
324 }
325
326 void MyGLCanvas::Render()
327 {
328     unsigned int i;

```

```

326 SetCurrent();
327 if (!init)
328 {
329     InitGL();
330     init = true;
331 }
332 glClear(GL_COLOR_BUFFER_BIT);
333 if (c->GetTotalCycles() > 0 && c->mmz()->moncount() > 0)
334 {
335     int totalCycles = c->GetTotalCycles();
336     // Scale traces (within limits) to fit the size of the canvas
337     int monCount = c->mmz()->moncount();
338     int canvasHeight = GetClientSize().GetHeight();
339     int canvasWidth = GetClientSize().GetWidth();
340     int spacing = (canvasHeight-10)/monCount;
341     if (spacing>200) spacing = 200;
342     if (spacing<50) spacing = 50;
343     int height = 0.8*(spacing-14);
344     int xOffset = maxMonNameWidth+10;
345     double xScale = double(canvasWidth-xOffset-10)/c->GetTotalCycles()
346     ();
347     if (xScale<minXScale) xScale = minXScale;
348     if (xScale>50) xScale = 50;
349
350     wxRect visibleRegion(0,0,canvasWidth,canvasHeight);
351
352     // Work out the best interval to use between displayed cycle
353     // numbers on the x axis
354     // Intervals will be 1, 2, or 5 * pow(10,n) cycles between
355     // displayed numbers, where n>=0
356     int minAxisLabelIntervalPixels = ceil(ceil(log10(totalCycles))
357     *8*2);
358     int base = 1;
359     int baseMultiples[] = {1,2,5};
360     int axisLabelInterval = 1;
361     while (axisLabelInterval<=totalCycles)
362     {
363         for (i=0; i<3; i++)
364         {
365             axisLabelInterval = baseMultiples[i]*base;
366             if (axisLabelInterval*xScale >= minAxisLabelIntervalPixels)
367                 break;
368         }
369         if (axisLabelInterval*xScale >= minAxisLabelIntervalPixels)
370             break;
371         base *= 10;
372     }
373     // Draw the traces
374     for (i=0; i<mons.size(); i++)
375     {

```

```

373     mons[i].SetGeometry(xOffset+scrollX, canvasHeight-scrollY-
374     spacing*i-spacing/2, xScale, height, spacing*0.075, spacing, 10,
375     axisLabelInterval);
376     mons[i].Draw(this, visibleRegion);
377     mons[i].DrawName(this, visibleRegion);
378     }
379     if (errorMessage != wxT(""))
380     {
381         DrawInfoTextCentre(errorMessage, true);
382     }
383     else if (errorMessage != wxT(""))
384     {
385         DrawInfoTextCentre(errorMessage, true);
386     }
387     else if (c->netz()->devicelist()==NULL)
388     {
389         DrawInfoTextCentre(_("No circuit loaded"));
390     }
391     else if (c->mmz()->moncount()==0)
392     {
393         DrawInfoTextCentre(_("No monitors"));
394     }
395     else
396     {
397         DrawInfoTextCentre(_("No simulation results. Use the run button."
398         ));
399     }
400     // We've been drawing to the back buffer, flush the graphics
401     // pipeline and swap the back buffer to the front
402     glFlush();
403     SwapBuffers();
404 }
405 void MyGLCanvas::SetErrorMessage(wxString txt)
406 {
407     // Set an error message to be displayed in the centre of the canvas
408     errorMessage = txt;
409     Redraw();
410 }
411 void MyGLCanvas::ClearErrorMessage()
412 {
413     errorMessage = wxT("");
414 }
415 void MyGLCanvas::DrawInfoTextCentre(wxString txt, bool isError)
416 {
417     // Draw a message in a box in the centre of the canvas
418     // isError makes it a red box
419

```

```

420 int canvasHeight = GetClientSize().GetHeight();
421 int canvasWidth = GetClientSize().GetWidth();
422 int textWidth = GetGlutTextWidth(txt);
423 wxRect background(canvasWidth/2-textWidth/2-15, canvasHeight/2-15,
    textWidth+30, 30);
424 if (isError) glColor4f(1.0, 0.85, 0.85, 0.95);
425 else glColor4f(0.85, 0.85, 1.0, 0.95);
426 glBegin(GL_QUADS);
427 wxRect_GlVertex(background);
428 glEnd();
429 if (isError) glColor4f(0.7, 0.0, 0.0, 0.95);
430 else glColor4f(0.0, 0.0, 0.7, 0.95);
431 glBegin(GL_LINE_LOOP);
432 wxRect_GlVertex(background);
433 glEnd();
434 glColor4f(0.0, 0.0, 0.0, 1.0);
435 DrawGlutText(canvasWidth/2-textWidth/2, canvasHeight/2-4, txt);
436 }
437
438 void MyGLCanvas::InitGL()
439 // Function to initialise the GL context
440 {
441     int w, h;
442
443     GetClientSize(&w, &h);
444     SetCurrent();
445     glDrawBuffer(GL_BACK);
446     glClearColor(1.0, 1.0, 1.0, 0.0);
447     glViewport(0, 0, (GLint) w, (GLint) h);
448     glMatrixMode(GL_PROJECTION);
449     glLoadIdentity();
450     glOrtho(0, w, 0, h, -1, 1);
451     glMatrixMode(GL_MODELVIEW);
452     glLoadIdentity();
453     glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
454     glEnable(GL_BLEND);
455 }
456
457 void MyGLCanvas::OnPaint(wxPaintEvent& event)
458 // Callback function for when the canvas is exposed
459 {
460     wxPaintDC dc(this); // required for correct refreshing under MS
    windows
461     Render();
462 }
463
464 void MyGLCanvas::OnSize(wxSizeEvent& event)
465 // Callback function for when the canvas is resized
466 {
467     wxGLCanvas::OnSize(event); // required on some platforms
468     init = false;

```

```

469 Redraw(); // required by some buggy nvidia graphics drivers,
    harmless on other platforms!
470 }
471
472 void MyGLCanvas::OnMouse(wxMouseEvent& event)
473 // Callback function for mouse events inside the GL canvas
474 {
475     wxString text;
476     int w, h;;
477
478     GetClientSize(&w, &h);
479     if (event.ButtonDown()) text.Printf(wxT("Mouse button %d pressed at
        %d %d"), event.GetButton(), event.m_x, h-event.m_y);
480     if (event.ButtonUp()) text.Printf(wxT("Mouse button %d released at
        %d %d"), event.GetButton(), event.m_x, h-event.m_y);
481     if (event.Dragging()) text.Printf(wxT("Mouse dragged to %d %d"),
        event.m_x, h-event.m_y);
482     if (event.Leaving()) text.Printf(wxT("Mouse left window at %d %d"),
        event.m_x, h-event.m_y);
483
484     //if (event.ButtonDown() || event.ButtonUp() || event.Dragging() ||
        event.Leaving()) Render(text);
485 }

```

### 3.2.5 GUI: miscellaneous widgets and functions

Listing 16: gui-misc.h

```

1 #ifndef gui_misc_h
2 #define gui_misc_h
3
4 #include <wx/combobox.h>
5 #include <wx/textctrl.h>
6 #include <wx/listbox.h>
7 #include <wx/checklst.h>
8 #include "circuit.h"
9
10 // This file contains misc widgets
11 // Also some functions for things done in lots of places in the GUI,
    like CircuitElementInfoVector_to_wxArrayString
12
13 // CheckListBox to control switch states
14 class SwitchesCheckListBox: public wxCheckListBox
15 {
16 public:
17     SwitchesCheckListBox(circuit* circ, wxWindow* parent, wxWindowID id
        = wxID_ANY, const wxPoint& pos = wxDefaultPosition, const wxSize
        & size = wxDefaultSize, long style = 0);

```

```

18 ~SwitchesCheckListBox();
19 void OnCircuitChanged();
20 private:
21     circuit* c;
22     CircuitElementInfoVector switches;
23     void OnSwitchChanged(wxCommandEvent& event);
24     DECLARE_EVENT_TABLE()
25 };
26
27 extern wxString devicekindstrings[baddevice];
28
29 // Dropdown box to select a devicekind, optionally filtering the
30 // displayed types to those given in filterDevicekinds
31 class DevicekindDropdown: public wxComboBox
32 {
33 public:
34     DevicekindDropdown(wxWindow* parent, wxWindowID id = wxID_ANY,
35         vector<devicekind> filterDevicekinds = vector<devicekind>());
36     devicekind GetDevicekind();
37     void SetDevicekind(devicekind dk);
38 private:
39     vector<devicekind> filter;
40 };
41
42 // Text control for entering a device name, with a function for
43 // validating the new name
44 class DeviceNameTextCtrl: public wxTextCtrl
45 {
46 public:
47     DeviceNameTextCtrl(wxWindow* parent, wxWindowID id = wxID_ANY,
48         const wxString& value = wxT(""));
49     // Validate the new name, optionally showing an error message
50     // allowExisting is a name that can be chosen even if a device
51     // already exists with the same name (so this control can be used
52     // for editing names of existing devices), use blankname to disallow
53     // any existing device name
54     bool CheckValid(circuit* c, name allowExisting=blankname, bool
55         errorDialog=true);
56 };
57
58 // List of devices (single selection, selection changes are handled
59 // by the supplied SelectedDevice object)
60 class DevicesListBox: public wxListBox
61 {
62 public:
63     DevicesListBox(circuit* circ, SelectedDevice* selectedDev_in,
64         wxWindow* parent, wxWindowID id);
65     ~DevicesListBox();
66     void OnCircuitChanged();
67     void OnDeviceSelectionChanged();
68     void ReleasePointers();

```

```

59     devlink GetSelectionAfterDelete();
60 private:
61     SelectedDevice* selectedDev;
62     circuit* c;
63     CircuitElementInfoVector devs;
64     bool isSetting;
65     void OnListBoxSelectionChanged(wxCommandEvent& event);
66     DECLARE_EVENT_TABLE()
67 };
68
69 // Show an error message box
70 void ShowErrorMsgDialog(wxWindow* parent, wxString txt);
71
72 // Put namestrs from a CircuitElementInfoVector into a wxArrayString
73 void CircuitElementInfoVector_to_wxArrayString(const
74     CircuitElementInfoVector& src, wxArrayString& dest);
75
76 #endif /* gui_misc_h */

```

Listing 17: gui-misc.cc

```

1 #include "gui-misc.h"
2 #include <wx/msgdlg.h>
3 #include <algorithm>
4 using namespace std;
5
6 SwitchesCheckListBox::SwitchesCheckListBox(circuit* circ, wxWindow*
7     parent, wxWindowID id, const wxPoint& pos, const wxSize& size,
8     long style)
9 : wxCheckListBox(parent, id, pos, size, 0, NULL, style & ~wxLB_SORT
10 )
11 {
12     c = circ;
13     if (c) c->circuitChanged.Attach(this, &SwitchesCheckListBox::
14         OnCircuitChanged);
15     OnCircuitChanged();
16 }
17
18 SwitchesCheckListBox::~SwitchesCheckListBox()
19 {
20     if (c) c->circuitChanged.Detach(this);
21 }
22
23 void SwitchesCheckListBox::OnCircuitChanged()
24 {
25     if (!c) return;
26
27     // Update switch names
28     switches.resize(0);
29     devlink d = c->netz()->devicelist();
30     while (d!=NULL)

```

```

27 {
28     if (d->kind == aswitch) switches.push_back(CircuitElementInfo(d))
29     ;
30     d = d->next;
31 }
32 switches.UpdateSignalNames(c);
33 wxString switchNames;
34 CircuitElementInfoVector_to_wxArrayString(switches, switchNames);
35 Set(switchNames);
36 // Update switch states
37 for (int i=0; i<switches.size(); i++)
38 {
39     Check(i, switches[i].d->swstate==high);
40 }
41
42 void SwitchesCheckListBox::OnSwitchChanged(wxCommandEvent& event)
43 {
44     if (!c) return;
45
46     // Checkbox changed, update underlying switch state
47     int i = event.GetInt();
48     if (i>=switches.size())
49     {
50         ShowErrorMsgDialog(this, _("Tried to change unknown switch"));
51         return;
52     }
53     if (IsChecked(i))
54         switches[i].d->swstate = high;
55     else
56         switches[i].d->swstate = low;
57
58     c->circuitChanged.Trigger();
59 }
60
61 BEGIN_EVENT_TABLE(SwitchesCheckListBox, wxCheckListBox)
62     EVT_CHECKLISTBOX(wxID_ANY, SwitchesCheckListBox::OnSwitchChanged)
63 END_EVENT_TABLE()
64
65 wxString devicekindstrings[baddevice] = {"Switch", "Clock", "AND gate", "NAND gate", "OR gate", "NOR gate", "XOR gate", "D-type flip-flop"};
66
67 DevicekindDropdown::DevicekindDropdown(wxWindow* parent, wxWindowID
68     id, vector<devicekind> filterDevicekinds) :
69     wxComboBox(parent, id, wxT(""), wxDefaultPosition, wxDefaultSize,
70     0, NULL, wxCB_READONLY)
71 {
72     filter = filterDevicekinds;

```

```

72 // Fill dropdown with available device kinds (optionally filtering
73 // to only include those listed in filterDevicekinds)
74 for (int i=0; i<baddevice; i++)
75 {
76     if (filter.size()==0 || find(filter.begin(), filter.end(),
77         devicekind(i)) != filter.end())
78         Append(devicekindstrings[i]);
79 }
80
81 devicekind DevicekindDropdown::GetDevicekind()
82 {
83     for (int i=0; i<baddevice; i++)
84     {
85         if (devicekindstrings[i] == GetValue())
86             return devicekind(i);
87     }
88     return baddevice;
89 }
90
91 void DevicekindDropdown::SetDevicekind(devicekind dk)
92 {
93     if (dk>=0 && dk<baddevice)
94     {
95         if (filter.size()==0 || find(filter.begin(), filter.end(), dk) !=
96             filter.end())
97             SetValue(devicekindstrings[dk]);
98     }
99 }
100
101 DeviceNameTextCtrl::DeviceNameTextCtrl(wxWindow* parent, wxWindowID
102     id, const wxString& value) :
103     wxTextCtrl(parent, id, value)
104 {}
105
106 bool DeviceNameTextCtrl::CheckValid(circuit* c, name allowExisting,
107     bool errorDialog)
108 {
109     if (!GetParent()) errorDialog = false;
110
111     if (GetValue() == wxT(""))
112     {
113         if (errorDialog) ShowErrorMsgDialog(GetParent(), _("Device name
114             cannot be blank"));
115         return false;
116     }
117     if (!isalpha(GetValue()[0]))
118     {
119         if (errorDialog) ShowErrorMsgDialog(GetParent(), _("Device name
120             must start with a letter"));

```

```

116     return false;
117 }
118 if (!circuit::IsDeviceNameValid(string(GetValue().mb_str())))
119 {
120     if (errorDialog) ShowErrorMsgDialog(GetParent(), _("Device name
must contain only letters and numbers"));
121     return false;
122 }
123 if (c)
124 {
125     name newname = c->nmz()->cvtname(string(GetValue().mb_str()));
126     if (newname!=blankname && newname<=lastreservedname)
127     {
128         if (errorDialog) ShowErrorMsgDialog(GetParent(), _("Device name
cannot be a reserved word"));
129         return false;
130     }
131     if (newname!=allowExisting && c->netz()->finddevice(newname)!=
NULL)
132     {
133         if (errorDialog) ShowErrorMsgDialog(GetParent(), _("A device
already exists with that name"));
134         return false;
135     }
136 }
137 return true;
138 }
139
140
141 DevicesListBox::DevicesListBox(circuit* circ, SelectedDevice*
selectedDev_in, wxWindow* parent, wxWindowID id) :
142     wxListBox(parent, id, wxDefaultPosition, wxDefaultSize, 0, NULL,
wxLB_NEEDED_SB | wxLB_SINGLE)
143 {
144     selectedDev = selectedDev_in;
145     c = circ;
146     isSetting = false;
147     c->circuitChanged.Attach(this, &DevicesListBox::OnCircuitChanged);
148     selectedDev->changed.Attach(this, &DevicesListBox::
OnDeviceSelectionChanged);
149     OnCircuitChanged();
150 }
151
152 DevicesListBox::~DevicesListBox()
153 {
154     ReleasePointers();
155 }
156
157 void DevicesListBox::ReleasePointers()
158 {
159     if (selectedDev)

```

```

160         selectedDev->changed.Detach(this);
161     selectedDev = NULL;
162     if (c)
163         c->circuitChanged.Detach(this);
164     c = NULL;
165 }
166
167 void DevicesListBox::OnCircuitChanged()
168 {
169     if (!c || !selectedDev) return;
170
171     devs.resize(0);
172     devs.push_back_all_devs(c->netz()->devicelist());
173     devs.UpdateSignalNames(c);
174     sort(devs.begin(), devs.end(), CircuitElementInfo_namestrcmp);
175
176     int selectedIndex = -1;
177     wxArrayString devNames;
178     devNames.Alloc(devs.size());
179     for (int i=0; i<devs.size(); i++)
180     {
181         if (devs[i].d==selectedDev->Get())
182             selectedIndex = i;
183         wxString desc(devs[i].namestr.c_str(), wxConvUTF8);
184         if (!c->dmz()->CheckDeviceInputs(devs[i].d))
185             desc = desc + wxT(" ") + _("(disconnected input)");
186         devNames.Add(desc);
187     }
188     Set(devNames);
189
190     if (selectedIndex<0)
191     {
192         if (devs.size()!=0)
193             selectedDev->Set(devs[0].d);
194         else
195             selectedDev->Set(NULL);
196     }
197     else
198     {
199         SetSelection(selectedIndex);
200     }
201 }
202
203 void DevicesListBox::OnDeviceSelectionChanged()
204 {
205     if (!selectedDev || isSetting) return;
206
207     for (int i=0; i<devs.size(); i++)
208     {
209         if (devs[i].d = selectedDev->Get())
210         {

```



```

211     SetSelection(i);
212     return;
213 }
214 }
215 SetSelection(wxNOT_FOUND);
216 }
217
218 void DevicesListBox::OnListBoxSelectionChanged(wxCommandEvent& event)
219 {
220     if (!selectedDev) return;
221
222     devlink d;
223     int i = GetSelection();
224     if (i>=0 && i<devs.size())
225     {
226         isSetting = true; // temporarily prevent DevicesListBox::
                             // OnDeviceSelectionChanged() from being run, otherwise the
                             // selection doesn't change properly
227         selectedDev->Set(devs[i].d);
228         isSetting = false;
229     }
230 }
231
232 devlink DevicesListBox::GetSelectionAfterDelete()
233 {
234     if (!c || !selectedDev || !selectedDev->Get())
235     {
236         if (devs.size()) return devs[0].d;
237         else return NULL;
238     }
239     if (devs.size()<=1) return NULL;
240
241     int i = GetSelection()+1;
242     if (i>=devs.size()) i = devs.size()-1;
243     if (i==GetSelection()) i = 0;
244     return devs[i].d;
245 }
246
247 BEGIN_EVENT_TABLE(DepicesListBox, wxListBox)
248     EVT_LISTBOX(wxID_ANY, DepicesListBox::OnListBoxSelectionChanged)
249 END_EVENT_TABLE()
250
251 void ShowErrorMsgDialog(wxWindow* parent, wxString txt)
252 {
253     wxMessageDialog dlg(parent, txt, _("Error"), wxCANCEL |
254         wxICON_ERROR);
255     dlg.ShowModal();
256 }
257

```

```

258 void CircuitElementInfoVector_to_wxArrayString(const
    CircuitElementInfoVector& src, wxString& dest)
259 {
260     dest.Clear();
261     dest.Alloc(src.size());
262     for (CircuitElementInfoVector::const_iterator it=src.begin(); it<
        src.end(); ++it)
263     {
264         dest.Add(wxString(it->namestr.c_str(), wxConvUTF8));
265     }
266 }

```

### 3.2.6 GUI: widget IDs

Listing 18: gui-id.h

```

1  #ifndef gui_id_h
2  #define gui_id_h
3
4  enum
5  {
6      MY_SPINCNTL_ID = wxID_HIGHEST + 1,
7      MY_TEXTCTRL_ID,
8      MY_BUTTON_ID,
9      MENU_CLEAR_CIRCUIT,
10     OUTPUT_TEXTCTRL_ID,
11     SIMCTRL_BUTTON_RUN_ID,
12     SIMCTRL_BUTTON_CONT_ID,
13     MONITORS_ADD_BUTTON_ID,
14     MONITORS_DEL_BUTTON_ID,
15     SWITCHES_CTRL_ID,
16     DEVICES_EDIT_BUTTON_ID,
17     DEVICES_APPLY_BUTTON_ID,
18     DEVICECREATE_BUTTON_ID,
19     DEVIDEDELETE_BUTTON_ID,
20     DEVICES_ADDCONN_BUTTON_ID,
21     DEVICES_DELCONN_BUTTON_ID,
22     DEVICENAME_TEXTCTRL_ID,
23     DEVICEOUTPUT_MONITOR_CB_ID
24 };
25
26 #endif /* gui_id_h */

```

### 3.2.7 Device editing GUI: dialogs

Listing 19: gui-devices.h

```

1 #ifndef gui_devices_h
2 #define gui_devices_h
3
4 #include "gui-devices-infopanel.h"
5 #include <wx/wx.h>
6 #include <wx/spinctrl.h>
7 #include <wx/listbox.h>
8 #include <wx/panel.h>
9 #include <wx/combobox.h>
10 #include "network.h"
11 #include "circuit.h"
12 #include "observer.h"
13 #include "gui-misc.h"
14 #include <string>
15 #include <vector>
16 using namespace std;
17
18 class NewDeviceDialog: public wxDialog
19 {
20 public:
21     NewDeviceDialog(circuit* circ, wxWindow* parent, wxWindowID id,
22         const wxString& title);
23     devlink newdev;
24 private:
25     circuit* c;
26     DevicekindDropdown* dkindDropdown;
27     DeviceNameTextCtrl* devicenameCtrl;
28     void OnOK(wxCommandEvent& event);
29     DECLARE_EVENT_TABLE()
30 };
31
32 class ChooseOutputDialog: public wxDialog
33 {
34 public:
35     ChooseOutputDialog(circuit* circ, wxWindow* parent, wxWindowID id,
36         const wxString& title, const wxString& description);
37     CircuitElementInfo result;
38 private:
39     circuit* c;
40     wxListBox* lbox;
41     CircuitElementInfoVector outputs;
42     void OnOK(wxCommandEvent& event);
43     DECLARE_EVENT_TABLE()
44 };
45
46 // For a given output, allow input(s) to be chosen to connect to it
47 // Makes the connection(s) when the OK button is clicked
48 class ConnectToOutputDialog: public wxDialog

```

```

48 {
49 public:
50     ConnectToOutputDialog(circuit* circ, outlink outp, wxWindow*
51         parent, wxWindowID id, const wxString& title);
52 private:
53     circuit* c;
54     outlink o;
55     CircuitElementInfoVector inputs;
56     wxListBox* lbox;
57     void OnOK(wxCommandEvent& event);
58     DECLARE_EVENT_TABLE()
59 };
60
61 class DevicesDialog: public wxDialog
62 {
63 public:
64     DevicesDialog(circuit* circ, wxWindow* parent, wxWindowID id, const
65         wxString& title, devlink d=NULL);
66     ~DevicesDialog();
67     void OnDeviceSelectionChanged();
68 private:
69     circuit* c;
70     SelectedDevice* selectedDev;
71     DevicesListBox* devListBox;
72     DeviceDetailsPanel* detailsPanel;
73     DeviceInputsPanel* inputsPanel;
74     vector<DeviceOutputPanel*> outputPanels;
75     wxBoxSizer* mainSizer;
76     wxBoxSizer* outputsSizer;
77     void DestroyDeviceWidgets();
78     void OnDeleteButton(wxCommandEvent& event);
79     void OnCreateButton(wxCommandEvent& event);
80     DECLARE_EVENT_TABLE()
81 };
82
83 #endif /* gui_devices_h */

```

Listing 20: gui-devices.cc

```

1 #include "gui-devices.h"
2 #include "gui-id.h"
3 #include "network.h"
4 #include "wx_icon.xpm"
5 #include <algorithm>
6 #include <climits>
7 using namespace std;
8
9 // The devices editing GUI

```



```

10 DevicesDialog::DevicesDialog(circuit* circ, wxWindow* parent,
    wxWindowID id, const wxString& title, devlink d) :
11 wxDialog(parent, id, title, wxDefaultPosition, wxDefaultSize,
    wxDEFAULT_DIALOG_STYLE | wxRESIZE_BORDER)
12 {
13     c = circ;
14     outputsSizer = NULL;
15     outputPanels.resize(0);
16     inputsPanel = NULL;
17     detailsPanel = NULL;
18     // A class to store a pointer to the currently selected device, and
    an ObserverSubject which is triggered when it changes
19     selectedDev = new SelectedDevice();
20     selectedDev->Set(d);
21     selectedDev->changed.Attach(this, &DevicesDialog::
        OnDeviceSelectionChanged);
22
23     wxBoxSizer* topsizer = new wxBoxSizer(wxHORIZONTAL);
24     wxBoxSizer* deviceListSizer = new wxBoxSizer(wxVERTICAL);
25     wxBoxSizer* ioSizer = new wxBoxSizer(wxHORIZONTAL);
26     mainSizer = new wxBoxSizer(wxVERTICAL);
27     // List of devices, which can be used to change the selected device
28     devListBox = new DevicesListBox(c, selectedDev, this, wxID_ANY);
29     devListBox->SetMinSize(wxSize(150,-1));
30     deviceListSizer->Add(devListBox, 1, wxALL | wxEXPAND, 10);
31     deviceListSizer->Add(new wxButton(this, DEVICES_CREATE_BUTTON_ID, _("
        Add device")), 0, (wxALL & ~wxTOP) | wxEXPAND, 10);
32
33     // Widgets to display device details
34     inputsPanel = new DeviceInputsPanel(c, selectedDev, this);
35     ioSizer->Add(inputsPanel, 1, wxEXPAND | wxALL, 5);
36     outputsSizer = new wxBoxSizer(wxVERTICAL);
37     ioSizer->Add(outputsSizer, 1, wxEXPAND | wxALL, 0);
38     mainSizer->Add(ioSizer, 1, wxEXPAND | wxALL, 5);
39
40     topsizer->Add(deviceListSizer, 0, wxEXPAND, 0);
41     topsizer->Add(mainSizer, 3, wxEXPAND, 0);
42
43     mainSizer->Add(CreateButtonSizer(wxOK), 0, wxALL | wxEXPAND, 10);
44     SetSizerAndFit(topsizer);
45     SetSizeHints(400,300);
46
47     OnDeviceSelectionChanged();
48 }
49
50 DevicesDialog::~DevicesDialog()
51 {
52     // detach all widgets from selectedDev->changed observer before
    deleting selectedDev
53     selectedDev->changed.Detach(this);
54     devListBox->ReleasePointers();

```

```

55     inputsPanel->ReleasePointers();
56     for (vector<DeviceOutputPanel*>::iterator it=outputPanels.begin();
        it<outputPanels.end(); ++it)
57     {
58         (*it)->ReleasePointers();
59     }
60     delete selectedDev;
61 }
62
63 void DevicesDialog::OnDeviceSelectionChanged()
64 {
65     // The widgets in the device details panel and the number of output
    details panels vary according to devicekind, so it's easier to
    just remove them all and recreate when a different device is
    selected
66
67     DestroyDeviceWidgets();
68
69     if (!selectedDev || !c || !outputsSizer) return;
70
71     devlink d = selectedDev->Get();
72
73     // Device details panel
74     detailsPanel = new DeviceDetailsPanel(c, selectedDev, this);
75     mainSizer->Insert(0, detailsPanel, 0, wxEXPAND | wxALL, 10);
76
77     if (selectedDev->Get())
78     {
79         // Make a sorted list of device outputs
80         CircuitElementInfoVector outputs;
81         outputs.push_back_dev_outputs(selectedDev->Get());
82         outputs.UpdateSignalNames(c);
83         sort(outputs.begin(), outputs.end(),
            CircuitElementInfo_namestrcmp);
84         // Create a DeviceOutputPanel for each
85         for (CircuitElementInfoVector::iterator it=outputs.begin(); it<
            outputs.end(); ++it)
86         {
87             DeviceOutputPanel* opanel = new DeviceOutputPanel(c, it->o,
                this);
88             outputPanels.push_back(opanel);
89             outputsSizer->Add(opanel, 1, wxEXPAND | wxALL, 5);
90         }
91     }
92
93     Layout();
94     Fit();
95 }
96
97 void DevicesDialog::DestroyDeviceWidgets()
98 {

```

```

99 // Remove the device details and output details panels
100 for (vector<DeviceOutputPanel*>::iterator it=outputPanels.begin();
101      it<outputPanels.end(); ++it)
102 {
103     (*it)->Destroy();
104 }
105 outputPanels.resize(0);
106 if (detailsPanel)
107 {
108     detailsPanel->Destroy();
109     detailsPanel = NULL;
110 }
111 }
112 void DevicesDialog::OnDeleteButton(wxCommandEvent& event)
113 {
114     if (!c || !selectedDev || !selectedDev->Get()) return;
115
116     devlink d = selectedDev->Get();
117     // Get the device which should be selected after the currently
118     // selected one is deleted (usually the one after it in the list),
119     // and select it before deleting the currently selected device
120     devlink newDev = devListBox->GetSelectionAfterDelete();
121     selectedDev->Set(newDev);
122     // Delete device (and related monitors)
123     c->RemoveDevice(d);
124 }
125 void DevicesDialog::OnCreateButton(wxCommandEvent& event)
126 {
127     if (!c || !selectedDev) return;
128     // Dialog to select new device type
129     NewDeviceDialog dlg(c, this, wxID_ANY, _("Create new device"));
130     if (dlg.ShowModal()==wxID_OK && dlg.newdev)
131     {
132         // Select the new device (created by dialog when OK is clicked)
133         selectedDev->Set(dlg.newdev);
134         c->circuitChanged.Trigger();
135     }
136 }
137 BEGIN_EVENT_TABLE(DevicesDialog, wxDialog)
138     EVT_BUTTON(DEVICEDELETE_BUTTON_ID, DevicesDialog::OnDeleteButton)
139     EVT_BUTTON(DEVICECREATE_BUTTON_ID, DevicesDialog::OnCreateButton)
140 END_EVENT_TABLE()
141
142 ChooseOutputDialog::ChooseOutputDialog(circuit* circ, wxWindow*
143     parent, wxWindowID id, const wxString& title, const wxString&
144     description):

```

```

144 wxDialog(parent, id, title, wxDefaultPosition, wxDefaultSize,
145     wxDEFAULT_DIALOG_STYLE | wxRESIZE_BORDER)
146 {
147     SetIcon(wxIcon(wx_icon));
148
149     c = circ;
150
151     // Make a list of all outputs in the circuit, for the listbox
152     outputs.push_back_all_outputs(c->netz()->devicelist());
153     outputs.UpdateSignalNames(c);
154     sort(outputs.begin(), outputs.end(), CircuitElementInfo_namestrcmp)
155     ;
156
157     wxArrayString displayedOutputs;
158     CircuitElementInfoVector_to_wxArrayString(outputs, displayedOutputs
159     );
160
161     // Create widgets
162     wxBoxSizer* topsizer = new wxBoxSizer(wxVERTICAL);
163     topsizer->Add(new wxStaticText(this, wxID_ANY, description), 0,
164         wxLEFT | wxRIGHT | wxTOP, 10);
165     lbox = new wxListBox(this, wxID_ANY, wxDefaultPosition,
166         wxDefaultSize, displayedOutputs, wxLB_SINGLE | wxLB_NEEDED_SB);
167     topsizer->Add(lbox, 1, wxALL | wxEXPAND, 10);
168     topsizer->Add(CreateButtonSizer(wxOK | wxCANCEL), 0, wxALL |
169         wxEXPAND, 10);
170     SetSizerAndFit(topsizer);
171 }
172 void ChooseOutputDialog::OnOK(wxCommandEvent& event)
173 {
174     int i = lbox->GetSelection();
175     if (i>=0 && i<outputs.size())
176     {
177         // Store a pointer to the selected output in a public member
178         // variable
179         result = outputs[i];
180     }
181     EndModal(wxID_OK);
182 }
183 BEGIN_EVENT_TABLE(ChooseOutputDialog, wxDialog)
184     EVT_BUTTON(wxID_OK, ChooseOutputDialog::OnOK)
185 END_EVENT_TABLE()
186
187 ConnectToOutputDialog::ConnectToOutputDialog(circuit* circ, outlink
188     outp, wxWindow* parent, wxWindowID id, const wxString& title):
189     wxDialog(parent, id, title, wxDefaultPosition, wxDefaultSize,
190         wxDEFAULT_DIALOG_STYLE | wxRESIZE_BORDER)
191 {

```

```

186 SetIcon(wxIcon(wx_icon));
187
188 c = circ;
189 o = outp;
190
191 // Make a list of all inputs in the circuit, for the listbox
192 inputs.push_back_all_inputs(c->netz()->devicelist());
193 inputs.UpdateSignalNames(c);
194 // Sort by whether the input is connected, then alphabetically by
   name
195 sort(inputs.begin(), inputs.end(),
   CircuitElementInfo_Iconnect_namestrcmp);
196
197 // Put the input names in the listbox, with a description of what (
   if anything) is connected to each one
198 wxArrayString lboxContents;
199 lboxContents.Alloc(inputs.size());
200 for (CircuitElementInfoVector::iterator it=inputs.begin(); it<
   inputs.end(); ++it)
201 {
202     wxString desc(it->namestr.c_str(), wxConvUTF8);
203     if (it->i->connect)
204     {
205         desc = desc + wxT(" (") + _("connected to ") + wxString(c->netz(
   )->getsignalstring(it->i->connect->dev->id, it->i->connect->id).
   c_str(), wxConvUTF8) + wxT(")");
206     }
207     else
208     {
209         desc = desc + _(" (unconnected)");
210     }
211     lboxContents.Add(desc);
212 }
213
214 // Create widgets
215 wxBoxSizer* topsizer = new wxBoxSizer(wxVERTICAL);
216 topsizer->Add(new wxStaticText(this, wxID_ANY, wxString(_("Choose
   input(s) to connect to ")) + wxString(c->netz()->getsignalstring(
   o->dev, o).c_str(), wxConvUTF8)), 0, wxLEFT | wxRIGHT | wxTOP,
   10);
217 lbox = new wxListBox(this, wxID_ANY, wxDefaultPosition,
   wxDefaultSize, lboxContents, wxLB_EXTENDED | wxLB_NEEDED_SB);
218 topsizer->Add(lbox, 1, wxALL | wxEXPAND, 10);
219 topsizer->Add(CreateButtonSizer(wxOK | wxCANCEL), 0, wxALL |
   wxEXPAND, 10);
220 SetSizerAndFit(topsizer);
221 }
222
223 void ConnectToOutputDialog::OnOK(wxCommandEvent& event)
224 {

```

```

225 // Connect all the selected inputs to the output that was passed in
   the constructor
226 wxArrayInt selections;
227 lbox->GetSelections(selections);
228 for (int i=0; i<selections.GetCount(); i++)
229 {
230     if (selections[i]<inputs.size())
231         inputs[selections[i]].i->connect = o;
232 }
233 c->circuitChanged.Trigger();
234 EndModal(wxID_OK);
235 }
236
237 BEGIN_EVENT_TABLE(ConnectToOutputDialog, wxDialog)
238 EVT_BUTTON(wxID_OK, ConnectToOutputDialog::OnOK)
239 END_EVENT_TABLE()
240
241
242 NewDeviceDialog::NewDeviceDialog(circuit* circ,wxWindow* parent,
   wxWindowID id, const wxString& title):
243     wxDialog(parent, id, title, wxDefaultPosition, wxDefaultSize,
   wxDEFAULT_DIALOG_STYLE | wxRESIZE_BORDER)
244 {
245     SetIcon(wxIcon(wx_icon));
246
247     c = circ;
248
249     wxBoxSizer* topsizer = new wxBoxSizer(wxVERTICAL);
250     // Device name textbox
251     topsizer->Add(new wxStaticText(this, wxID_ANY, _("New device name:"
   )), 0, wxLEFT | wxRIGHT | wxTOP, 10);
252     devicenameCtrl = new DeviceNameTextCtrl(this);
253     topsizer->Add(devicenameCtrl, 0, wxALL | wxEXPAND, 10);
254     // Device type dropdown
255     topsizer->Add(new wxStaticText(this, wxID_ANY, _("Device type:")),
   0, wxLEFT | wxRIGHT | wxTOP, 10);
256     dkindDropdown = new DevicekindDropdown(this);
257     dkindDropdown->SetSelection(0); // Select a default device type
258     topsizer->Add(dkindDropdown, 0, wxALL | wxEXPAND, 10);
259     topsizer->Add(CreateButtonSizer(wxOK | wxCANCEL), 0, wxALL |
   wxEXPAND, 10);
260     SetSizerAndFit(topsizer);
261 }
262
263 void NewDeviceDialog::OnOK(wxCommandEvent& event)
264 {
265     if (!devicenameCtrl->CheckValid(c, blankname, true))
266         return;
267
268     // Create a new device of the specified type, with some defaults
   for any options (like gate input count)

```

```

269 // Options can be changed immediately after creation in the device
    editing GUI
270 devicekind dk = dkindDropdown->GetDevicekind();
271 name devname = c->nmz()->lookup(string(devicenameCtrl->GetValue().
    mb_str()));
272 bool ok;
273 if (dk==aswitch)
274     c->dmz()->makedevice(dk, devname, 1, ok);
275 else if (dk==aclock)
276     c->dmz()->makedevice(dk, devname, 5, ok);
277 else
278     c->dmz()->makedevice(dk, devname, 2, ok);
279
280 // Put a pointer to the newly created device in a public member
    variable
281 newdev = c->netz()->finddevice(devname);
282 EndModal(wxID_OK);
283 }
284
285 BEGIN_EVENT_TABLE(NewDeviceDialog, wxDialog)
286     EVT_BUTTON(wxID_OK, NewDeviceDialog::OnOK)
287 END_EVENT_TABLE()

```

### 3.2.8 Device editing GUI: information panels

Listing 21: gui-devices-infopanel.h

```

1 #ifndef gui_devices_infopanel_h
2 #define gui_devices_infopanel_h
3
4 #include "circuit.h"
5 #include "observer.h"
6 #include "gui-misc.h"
7 #include <wx/panel.h>
8 #include <wx/gbsizer.h>
9 #include <wx/combobox.h>
10 #include <wx/textctrl.h>
11 #include <wx/stattext.h>
12 #include <wx/spinctrl.h>
13 #include <wx/checkbox.h>
14 #include <wx/button.h>
15
16 // Panel for display and modification of device properties
17 class DeviceDetailsPanel: public wxPanel
18 {
19 public:
20     DeviceDetailsPanel(circuit* circ, SelectedDevice* selectedDev_in,
        wxWindow* parent, wxWindowID id = wxID_ANY);

```

```

21 protected:
22     virtual void UpdateApplyButtonState_ExtraFields() {};
23     virtual void OnApply_ExtraFields(bool& changedSomething) {};
24     SelectedDevice* selectedDev;
25     circuit* c;
26     wxStaticBoxSizer* mainSizer;
27     wxGridBagSizer* gridsizer;
28     DeviceNameTextCtrl* devicenameCtrl;
29     wxButton* updateBtn;
30     wxStaticText* devicekindStaticText;
31
32     wxSpinCtrl* spinCtrl;
33     DevicekindDropdown* gateTypeDropdown;
34
35     void UpdateApplyButtonState();
36     void OnInputChanged(wxCommandEvent& event);
37     void OnApply(wxCommandEvent& event);
38     void ShowErrorMsg(wxString txt);
39     DECLARE_EVENT_TABLE()
40 };
41
42 // List of all inputs on a device and which output each is connected
    to, with buttons to add/remove connections
43 class DeviceInputsPanel: public wxPanel
44 {
45 public:
46     DeviceInputsPanel(circuit* circ, SelectedDevice* selectedDev_in,
        wxWindow* parent, wxWindowID id = wxID_ANY);
47     ~DeviceInputsPanel();
48     void ReleasePointers();
49 private:
50     SelectedDevice* selectedDev;
51     circuit* c;
52     CircuitElementInfoVector inps;
53     wxListBox* lbox;
54     wxButton *btnAddConn, *btnDelConn;
55     void UpdateInps();
56     void OnConnectButton(wxCommandEvent& event);
57     void OnDisconnectButton(wxCommandEvent& event);
58     void OnLBoxSelectionChanged(wxCommandEvent& event);
59     void UpdateControlStates();
60     void OnDeviceSelectionChanged();
61     DECLARE_EVENT_TABLE()
62 };
63
64 // Lists the inputs of other devices that a particular device output
    is connected to, with buttons to add/remove connections
65 class DeviceOutputPanel: public wxPanel
66 {
67 public:

```

```

68 DeviceOutputPanel(circuit* circ, outlink targetOutp, wxWindow*
69     parent, wxWindowID id = wxID_ANY);
69 ~DeviceOutputPanel();
70 void ReleasePointers();
71 private:
72     circuit* c;
73     outlink outp;
74     CircuitElementInfoVector inps;
75     wxListBox* lbox;
76     wxButton *btnAddConn, *btnDelConn;
77     wxCheckBox* monitorCheckbox;
78     void UpdateInps();
79     void OnConnectButton(wxCommandEvent& event);
80     void OnDisconnectButton(wxCommandEvent& event);
81     void OnLBoxSelectionChanged(wxCommandEvent& event);
82     void OnMonitorCheckboxChanged(wxCommandEvent& event);
83     void UpdateControlStates();
84     void OnDeviceSelectionChanged();
85     void OnMonitorsChanged();
86     DECLARE_EVENT_TABLE()
87 };
88
89 #endif /* gui_devices_infopanel_h */

```

Listing 22: gui-devices-infopanel.cc

```

1 #include "gui-devices-infopanel.h"
2 #include "gui-devices.h"
3 #include "gui-id.h"
4 #include <algorithm>
5
6 using namespace std;
7
8 DeviceInputsPanel::DeviceInputsPanel(circuit* circ, SelectedDevice*
9     selectedDev_in, wxWindow* parent, wxWindowID id) :
10     wxPanel(parent, id)
11 {
12     selectedDev = selectedDev_in;
13     c = circ;
14
15     wxStaticBoxSizer* mainSizer = new wxStaticBoxSizer(wxVERTICAL, this
16         , _("Inputs"));
17     // Listbox to show all inputs for this device
18     lbox = new wxListBox(this, wxID_ANY, wxDefaultPosition,
19         wxDefaultSize, 0, NULL, wxLB_NEEDED_SB | wxLB_EXTENDED);
20     lbox->SetMinSize(wxSize(150,100));
21     mainSizer->Add(lbox, 1, wxEXPAND | (wxALL & ~wxBOTTOM), 10);
22     // Connect and disconnect buttons
23     wxBoxSizer* buttonsSizer = new wxBoxSizer(wxHORIZONTAL);
24     btnAddConn = new wxButton(this, DEVICES_ADDCONN_BUTTON_ID, _("
25         Connect"));

```

```

22     btnDelConn = new wxButton(this, DEVICES_DELCONN_BUTTON_ID, _("
23         Disconnect"));
24     buttonsSizer->Add(btnAddConn, 1, wxEXPAND, 10);
25     buttonsSizer->Add(btnDelConn, 1, wxEXPAND, 10);
26     mainSizer->Add(buttonsSizer, 0, wxEXPAND | (wxALL & ~wxTOP) , 10);
27     SetSizerAndFit(mainSizer);
28
29     // Listen for device selection changes, and changes to device
30     details
31     selectedDev->changed.Attach(this, &DeviceInputsPanel::
32         OnDeviceSelectionChanged);
33     c->circuitChanged.Attach(this, &DeviceInputsPanel::UpdateInps);
34
35     UpdateInps();
36     UpdateControlStates();
37 }
38
39 DeviceInputsPanel::~DeviceInputsPanel()
40 {
41     ReleasePointers();
42 }
43
44 void DeviceInputsPanel::ReleasePointers()
45 {
46     if (selectedDev)
47         selectedDev->changed.Detach(this);
48     selectedDev = NULL;
49     if (c)
50         c->circuitChanged.Detach(this);
51     c = NULL;
52 }
53
54 void DeviceInputsPanel::UpdateInps()
55 {
56     wxArrayInt selections;
57     lbox->GetSelections(selections);
58
59     inps.resize(0);
60     wxArrayString lboxData;
61     if (selectedDev && c && selectedDev->Get())
62     {
63         // Make a list of all inputs for the selected device
64         inps.push_back_dev_inputs(selectedDev->Get());
65         // Reverse it, since creating from I1..I16 means ilist points to
66         // the last one, I16
67         reverse(inps.begin(), inps.end());
68         // Put input names in the listbox, with a description of what (if
69         // anything) is connected to each one
70         lboxData.Alloc(inps.size());
71         for (CircuitElementInfoVector::iterator it=inps.begin(); it<inps.
72             end(); ++it)

```

```

67     {
68         inplink i = it->i;
69         wxString desc(c->nmz()->getnamestring(i->id).c_str(),
70 wxConvUTF8);
71         if (i->connect)
72         {
73             desc = desc + wxT(" ") + _("connected to ") + wxString(c->
74 netz()->getsignalstring(i->connect->dev->id, i->connect->id).
75 c_str(), wxConvUTF8) + wxT(" ");
76         }
77         else
78         {
79             desc = desc + _(" (unconnected)");
80         }
81         lboxData.Add(desc);
82     }
83     lbox->Set(lboxData);
84     UpdateControlStates();
85 }
86 void DeviceInputsPanel::OnDeviceSelectionChanged()
87 {
88     lbox->SetSelection(wxNOT_FOUND);
89     if (c && selectedDev && selectedDev->Get())
90     {
91         // Selected device changed, update the displayed inputs
92         UpdateInps();
93         Show();
94     }
95     else
96     {
97         // Hide the device inputs details panel if no device is selected
98         Hide();
99     }
100 }
101 void DeviceInputsPanel::OnLBoxSelectionChanged(wxCommandEvent& event)
102 {
103     UpdateControlStates();
104 }
105 void DeviceInputsPanel::UpdateControlStates()
106 {
107     // Enable or disable add+remove connection buttons depending on
108     // whether any inputs are selected
109     wxArrayInt selections;
110     lbox->GetSelections(selections);
111     if (selections.GetCount())
112     {
113

```

```

114         btnAddConn->Enable();
115         btnDelConn->Enable();
116     }
117     else
118     {
119         btnAddConn->Disable();
120         btnDelConn->Disable();
121     }
122 }
123 void DeviceInputsPanel::OnConnectButton(wxCommandEvent& event)
124 {
125     wxArrayInt selections;
126     lbox->GetSelections(selections);
127     ChooseOutputDialog dlg(c, this, wxID_ANY, _("Choose output"),
128 wxPLURAL("Choose an output to connect this input to:", "Choose an
129 output to connect these inputs to:", selections.GetCount()));
130     if (dlg.ShowModal()==wxID_OK && dlg.result.o)
131     {
132         wxArrayInt selections;
133         lbox->GetSelections(selections);
134         // Connect all selected inputs to the output chosen in the dialog
135         for (int i=0; i<selections.GetCount(); i++)
136         {
137             if (selections[i]<inps.size())
138             {
139                 inps[selections[i]].i->connect = dlg.result.o;
140             }
141         }
142         c->circuitChanged.Trigger();
143     }
144 }
145 void DeviceInputsPanel::OnDisconnectButton(wxCommandEvent& event)
146 {
147     wxArrayInt selections;
148     lbox->GetSelections(selections);
149     // Disconnect all selected inputs from whatever they are connected
150     // to
151     for (int i=0; i<selections.GetCount(); i++)
152     {
153         if (i<inps.size()) inps[selections[i]].i->connect = NULL;
154     }
155     c->circuitChanged.Trigger();
156 }
157 BEGIN_EVENT_TABLE(DeviceInputsPanel, wxPanel)
158     EVT_LISTBOX(wxID_ANY, DeviceInputsPanel::OnLBoxSelectionChanged)
159     EVT_BUTTON(DEVICES_ADDCONN_BUTTON_ID, DeviceInputsPanel::
        OnConnectButton)

```



```

160 EVT_BUTTON(DEVICES_DELCONN_BUTTON_ID, DeviceInputsPanel::
    OnDisconnectButton)
161 END_EVENT_TABLE()
162
163 DeviceOutputPanel::DeviceOutputPanel(circuit* circ, outlink
164 targetOutp, wxWindow* parent, wxWindowID id) :
    wxPanel(parent, id)
165 {
166     outp = targetOutp;
167     c = circ;
168
169     wxString title = _("Output");
170     if (outp->id != blankname)
171         title = title + wxT(" ") + wxString(c->nmz()->getnamestring(outp
172 ->id).c_str(),wxConvUTF8);
173     wxStaticBoxSizer* mainSizer = new wxStaticBoxSizer(wxVERTICAL, this
174 , title);
175     // Checkbox to add/remove a monitor on this output
176     monitorCheckbox = new wxCheckBox(this, DEVICEOUTPUT_MONITOR_CB_ID,
177 _("Monitored"));
178     mainSizer->Add(monitorCheckbox, 0, wxALL, 10);
179     // Listbox showing all connected inputs
180     mainSizer->Add(new wxStaticText(this, wxID_ANY, _("Inputs connected
181 to this output:")), 0, wxLEFT | wxRIGHT, 10);
182     lbox = new wxListBox(this, wxID_ANY, wxDefaultPosition,
183 wxDefaultSize, 0, NULL, wxLB_NEEDED_SB | wxLB_EXTENDED);
184     lbox->SetMinSize(wxSize(150,50));
185     mainSizer->Add(lbox, 1, wxEXPAND | wxLEFT | wxRIGHT, 10);
186     // Connect and disconnect buttons
187     wxBoxSizer* buttonsSizer = new wxBoxSizer(wxHORIZONTAL);
188     btnAddConn = new wxButton(this, DEVICES_ADDCONN_BUTTON_ID, _("
189 Connect"));
190     btnDelConn = new wxButton(this, DEVICES_DELCONN_BUTTON_ID, _("
191 Disconnect"));
192     buttonsSizer->Add(btnAddConn, 1, wxEXPAND, 10);
193     buttonsSizer->Add(btnDelConn, 1, wxEXPAND, 10);
194     mainSizer->Add(buttonsSizer, 0, wxEXPAND | (wxALL & ~wxTOP) , 10);
195     SetSizerAndFit(mainSizer);
196
197     // Listen for changes to device details and monitors
198     c->circuitChanged.Attach(this, &DeviceOutputPanel::UpdateInps);
199     c->monitorsChanged.Attach(this, &DeviceOutputPanel::
200 OnMonitorsChanged);
201
202     UpdateInps();
203     UpdateControlStates();
204     OnMonitorsChanged();
205 }
206 DeviceOutputPanel::~DeviceOutputPanel()

```

```

201 {
202     ReleasePointers();
203 }
204
205 void DeviceOutputPanel::ReleasePointers()
206 {
207     if (c)
208     {
209         c->circuitChanged.Detach(this);
210         c->monitorsChanged.Detach(this);
211     }
212     c = NULL;
213 }
214
215 void DeviceOutputPanel::OnMonitorsChanged()
216 {
217     // If monitors are added or removed, update the value of the
218     // checkbox indicating whether this output is monitored
219     monitorCheckbox->SetValue(c->mmz()->IsMonitored(outp));
220 }
221
222 void DeviceOutputPanel::OnMonitorCheckboxChanged(wxCommandEvent&
223 event)
224 {
225     bool ok = false;
226     // Monitor checkbox changed, add or remove a monitor for this
227     // device
228     if (monitorCheckbox->IsChecked())
229     {
230         c->mmz()->makemonitor(outp->dev->id, outp->id, ok);
231         if (c->GetTotalCycles()!=0)
232             cout << wxString(_("Monitor added, run simulation again to see
233 updated signals")).mb_str() << endl;
234     }
235     else
236     {
237         c->mmz()->remmonitor(outp->dev->id, outp->id, ok);
238     }
239     if (ok) c->monitorsChanged.Trigger();
240 }
241
242 void DeviceOutputPanel::UpdateInps()
243 {
244     wxArrayInt selections;
245     lbox->GetSelections(selections);
246     inps.resize(0);
247     wxArrayString lboxData;
248     if (c)
249     {
250         // Make a list of all inputs connected to this output
251         devlink d = c->netz()->devicelist();

```

```

248 while (d != NULL)
249 {
250     inplink i = d->ilist;
251     while (i != NULL)
252     {
253         if (i->connect == outp) inps.push_back(CircuitElementInfo(d,i
254     ));
255         i = i->next;
256     }
257     d = d->next;
258 }
259 inps.UpdateSignalNames(c);
260 sort(inps.begin(), inps.end(), CircuitElementInfo_namestrcmp);
261 // Put the names in the listbox
262 lboxData.Alloc(inps.size());
263 for (CircuitElementInfoVector::iterator it=inps.begin(); it<inps.
264 end(); ++it)
265 {
266     lboxData.Add(wxString(it->namestr.c_str(), wxConvUTF8));
267 }
268 lbox->Set(lboxData);
269 UpdateControlStates();
270 }
271 void DeviceOutputPanel::OnListBoxSelectionChanged(wxCommandEvent& event)
272 {
273     UpdateControlStates();
274 }
275 void DeviceOutputPanel::UpdateControlStates()
276 {
277     // Enable or disable "remove connection" button depending on
278     // whether any inputs are currently selected
279     wxArrayInt selections;
280     lbox->GetSelections(selections);
281     if (selections.GetCount())
282     {
283         btnDelConn->Enable();
284     }
285     else
286     {
287         btnDelConn->Disable();
288     }
289 }
290 void DeviceOutputPanel::OnConnectButton(wxCommandEvent& event)
291 {
292     // Dialog to choose some inputs to connect to this output
293     ConnectToOutputDialog dlg(c, outp, this, wxID_ANY, _("Choose inputs
294     "));

```

```

295     dlg.ShowModal();
296 }
297 void DeviceOutputPanel::OnDisconnectButton(wxCommandEvent& event)
298 {
299     // Disconnect all selected inputs
300     wxArrayInt selections;
301     lbox->GetSelections(selections);
302     for (int i=0; i<selections.GetCount(); i++)
303     {
304         if (i<inps.size()) inps[selections[i]].i->connect = NULL;
305     }
306     c->circuitChanged.Trigger();
307 }
308 BEGIN_EVENT_TABLE(DeviceOutputPanel, wxPanel)
309 EVT_LISTBOX(wxID_ANY, DeviceOutputPanel::OnListBoxSelectionChanged)
310 EVT_CHECKBOX(DEVICEOUTPUT_MONITOR_CB_ID, DeviceOutputPanel::
311     OnMonitorCheckboxChanged)
312 EVT_BUTTON(DEVICES_ADDCONN_BUTTON_ID, DeviceOutputPanel::
313     OnConnectButton)
314 EVT_BUTTON(DEVICES_DELCONN_BUTTON_ID, DeviceOutputPanel::
315     OnDisconnectButton)
316 END_EVENT_TABLE()
317 DeviceDetailsPanel::DeviceDetailsPanel(circuit* circ, SelectedDevice*
318     selectedDev_in, wxWindow* parent, wxWindowID id) :
319     wxPanel(parent, id)
320 {
321     selectedDev = selectedDev_in;
322     c = circ;
323     mainSizer = new wxStaticBoxSizer(wxVERTICAL, this, _("Device
324     details"));
325     devicekind dk = baddevice;
326     devlink d = selectedDev->Get();
327     if (d)
328     {
329         dk = d->kind;
330         wxBoxSizer* buttonsSizer = new wxBoxSizer(wxHORIZONTAL);
331         gridsizer = new wxGridBagSizer();
332         wxString kindtext = wxT("Unknown device");
333         if (dk>=0 && dk<baddevice)
334             kindtext = devicekindstrings[dk];
335         // Device type text
336         gridsizer->Add(new wxStaticText(this, wxID_ANY, _("Device type:"
337         ), wxGBPosition(0,0), wxDefaultSpan, (wxALL & ~wxRIGHT) |
338         wxALIGN_CENTER_VERTICAL, 10);

```



```

339 devicekindStaticText = new wxStaticText(this, wxID_ANY, kindtext)
340 ;
341 gridSizer->Add(devicekindStaticText, wxGBPosition(0,1),
342 wxDefaultSpan, wxALL | wxALIGN_CENTER_VERTICAL, 10);
343 // Device name textbox
344 gridSizer->Add(new wxStaticText(this, wxID_ANY, _("Name:")),
345 wxGBPosition(0,3), wxDefaultSpan, (wxALL & ~wxRIGHT) |
346 wxALIGN_CENTER_VERTICAL, 10);
347 devicenameCtrl = new DeviceNameTextCtrl(this,
348 DEVICENAME_TEXTCTRL_ID, wxString(c->nmz()->getnamestring(d->id).
349 c_str(), wxConvUTF8));
350 gridSizer->Add(devicenameCtrl, wxGBPosition(0,4), wxDefaultSpan,
351 wxALL | wxEXPAND | wxALIGN_CENTER_VERTICAL, 10);
352 gridSizer->AddGrowableCol(1,3);
353 gridSizer->AddGrowableCol(4,5);
354
355 // Widgets specific to particular devices
356 if (d->kind == aclock)
357 {
358     // Half time period ("frequency" in existing code)
359     spinCtrl = new wxSpinCtrl(this);
360     spinCtrl->SetValue(selectedDev->Get()->frequency);
361     spinCtrl->SetRange(1,INT_MAX);
362     gridSizer->Add(new wxStaticText(this, wxID_ANY, _("Half-time-
363 period:")), wxGBPosition(1,3), wxDefaultSpan, wxLEFT | wxBOTTOM |
364 wxALIGN_CENTER_VERTICAL, 10);
365     gridSizer->Add(spinCtrl, wxGBPosition(1,4), wxDefaultSpan, (
366 wxALL & ~wxTOP) | wxEXPAND | wxALIGN_CENTER_VERTICAL, 10);
367 }
368 else if (d->kind == andgate || d->kind == nandgate || d->kind ==
369 orgate || d->kind == norgate)
370 {
371     // Gate type dropdown (allow changes of device kind between and
372 /nand/or/nor, since they have identical parameters and input/
373 output rules)
374     vector<devicekind> gateTypeChoices;
375     gateTypeChoices.push_back(andgate);
376     gateTypeChoices.push_back(nandgate);
377     gateTypeChoices.push_back(orgate);
378     gateTypeChoices.push_back(norgate);
379     gateTypeDropdown = new DevicekindDropdown(this, wxID_ANY,
380 gateTypeChoices);
381     gateTypeDropdown->SetDevicekind(d->kind);
382     gridSizer->Add(new wxStaticText(this, wxID_ANY, _("Gate type:"))
383 , wxGBPosition(1,0), wxDefaultSpan, wxLEFT | wxBOTTOM |
384 wxALIGN_CENTER_VERTICAL, 10);
385     gridSizer->Add(gateTypeDropdown, wxGBPosition(1,1),
386 wxDefaultSpan, (wxALL & ~wxTOP) | wxEXPAND |
387 wxALIGN_CENTER_VERTICAL, 10);
388 // Spinner to select number of inputs
389 spinCtrl = new wxSpinCtrl(this);

```

```

372 spinCtrl->SetValue(GetLinkedListLength(selectedDev->Get()->
373 ilist));
374 spinCtrl->SetRange(1,16);
375 gridSizer->Add(new wxStaticText(this, wxID_ANY, _("Inputs:")),
376 wxGBPosition(1,3), wxDefaultSpan, wxLEFT | wxBOTTOM |
377 wxALIGN_CENTER_VERTICAL, 10);
378 gridSizer->Add(spinCtrl, wxGBPosition(1,4), wxDefaultSpan, (
379 wxALL & ~wxTOP) | wxEXPAND | wxALIGN_CENTER_VERTICAL, 10);
380 }
381 mainSizer->Add(gridSizer, 0, wxEXPAND);
382
383 // Delete device button, and button for applying changes to
384 device details
385 buttonsSizer->AddStretchSpacer();
386 buttonsSizer->Add(new wxButton(this, DELETEDevice_BUTTON_ID, _("
387 Delete device")), 0, (wxALL & ~wxLEFT) | wxEXPAND, 10);
388 updateBtn = new wxButton(this, DEVICES_APPLY_BUTTON_ID, _("Apply
389 changes"));
390 updateBtn->Disable();
391 buttonsSizer->Add(updateBtn, 0, (wxALL & ~wxLEFT) | wxEXPAND, 10)
392 ;
393 mainSizer->Add(buttonsSizer, 0, wxEXPAND);
394 }
395 else
396 {
397     mainSizer->Add(new wxStaticText(this, wxID_ANY, _("No device
398 selected")));
399 }
400
401 SetSizerAndFit(mainSizer);
402 }
403
404 void DeviceDetailsPanel::OnApply(wxCommandEvent& event)
405 {
406     if (!c || !selectedDev || !selectedDev->Get()) return;
407
408     devlink d = selectedDev->Get();
409     bool changedSomething = false;
410
411     if (wxString(c->nmz()->getnamestring(d->id).c_str(), wxConvUTF8) !=
412 devicenameCtrl->GetValue())
413     {
414         // Update device name
415         if (devicenameCtrl->CheckValid(c, d->id, true))
416         {
417             d->id = c->nmz()->lookup(string(devicenameCtrl->GetValue().
418 mb_str()));
419             changedSomething = true;
420         }
421     }

```

```

412 if (d->kind == aclock)
413 {
414     // Update half-time-period
415     if (selectedDev->Get()->frequency != spinCtrl->GetValue())
416     {
417         selectedDev->Get()->frequency = spinCtrl->GetValue();
418         changedSomething = true;
419     }
420 }
421 else if (d->kind == andgate || d->kind == nandgate || d->kind ==
422         orgate || d->kind == norgate)
423 {
424     if (GetLinkedListLength(selectedDev->Get()->ilist) != spinCtrl->
425         GetValue())
426     {
427         // Change number of gate inputs
428         c->dmz()->SetGateInputCount(selectedDev->Get(), spinCtrl->
429             GetValue());
430         changedSomething = true;
431     }
432     if (selectedDev->Get()->kind != gateTypeDropdown->GetDevicekind()
433 )
434     {
435         // Change device type between and/nand/or/nor
436         selectedDev->Get()->kind = gateTypeDropdown->GetDevicekind();
437         devicekindStaticText->SetLabel(gateTypeDropdown->GetValue());
438         changedSomething = true;
439     }
440 }
441 if (changedSomething)
442 {
443     // Notify other bits of the GUI of any changes
444     c->circuitChanged.Trigger();
445     c->monitorsChanged.Trigger();
446     UpdateApplyButtonState();
447 }
448 }
449 void DeviceDetailsPanel::UpdateApplyButtonState()
450 {
451     // Update "apply changes" button state - only enable if field
452     // values are different to current device properties
453     updateBtn->Disable();
454     if (c && selectedDev && selectedDev->Get())
455     {
456         devlink d = selectedDev->Get();
457
458         if (wxString(c->nmz()->getnamestring(d->id).c_str(), wxConvUTF8)
459             != devicenameCtrl->GetValue())
460             updateBtn->Enable();
461     }
462 }

```

```

457 if (d->kind == aclock)
458 {
459     if (d->frequency != spinCtrl->GetValue())
460         updateBtn->Enable();
461 }
462 else if (d->kind == andgate || d->kind == nandgate || d->kind ==
463         orgate || d->kind == norgate)
464 {
465     if (GetLinkedListLength(d->ilist) != spinCtrl->GetValue())
466         updateBtn->Enable();
467     if (d->kind != gateTypeDropdown->GetDevicekind())
468         updateBtn->Enable();
469 }
470 }
471 }
472 void DeviceDetailsPanel::OnInputChanged(wxCommandEvent& event)
473 {
474     UpdateApplyButtonState();
475 }
476 BEGIN_EVENT_TABLE(DeviceDetailsPanel, wxPanel)
477     EVT_TEXT(wxID_ANY, DeviceDetailsPanel::OnInputChanged)
478     EVT_BUTTON(DEVICES_APPLY_BUTTON_ID, DeviceDetailsPanel::OnApply)
479 END_EVENT_TABLE()

```