IIA GF2 Software: Final Report

Jamie Magee (jam96)
Team 8
Gonville & Caius College

Contents

1 Description

Our logic simulator is able to simulate any number of circuits which include the following devices:

- Clocks
- Switches
- AND gates (Up to 16 inputs)
- NAND gates (Up to 16 inputs)
- OR gates (Up to 16 inputs)
- NOR gates (Up to 16 inputs)
- XOR gates
- D-Type flip-flops
- Signal generators

Figure 1: UML Class diagram of our logic simulator

2 Development Style

We split the development of our logic simulator into five major phases: specification, design, implementation, testing and maintenance. The timeframe was then decided for each task and each task was assigned to either a team member or the whole team, depending on the nature of the task.

Each member of the team was also assigned a general project role as follows:

Project manager: (T Hillel) - Responsible for project planning including delegation of tasks and ensuring that the project runs to the set timescale.

Programming administrator: (J Magee) - Responsible for upkeep of the project directory including performing builds and keeping legacy versions of the simulator.

Client representative: (M Jackson) - Responsible for ensuring that the project meets the client's requirements for the logic simulator as defined in Appendix A of the GF2 Project Handout.

We made significant use of git for revision control, as well as GitHub for tracking bugs in the software and features required by the client.

3 My Contribution

I took on responsibility for writing the names and scanner classes. In addition, I wrote approximately 25% of the parser class. Once the majority of the software was written, I designed a definition file for each error and warning our logic simulator can throw and then wrote a shell script which would attempt to run each definition file, and record the output from our logic simulator. The shell script can be found in code listing 6

4 Testing

We used two main tests of testing - unit and system testing - both of which are industry standard practices. For our unit testing, Martin wrote an errors class which compared the actual output from various units of code, to the expected output. For system testing I wrote a shell script which passed definition files to the logic simulator and recorded the output in a text file. There were two variatons on the shell script: One which ran known good definition files and therefore had to input the commands to run the simulation in addition to recording the output; Another which ran known bad definition files and only expected parsing errors which it recorded.

5 Conclusions

A Code Listings

A.1 Names Class

A.1.1 names.h

```
#ifndef names_h
#define names_h

#include <string>
#include <vector>

using namespace std;

//const int maxnames  = 200;  /* max number of distinct names */
//const int maxlength = 8;    /* max chars in a name string   */
const int blankname = -1;   /* special name                 */

const int lastreservedname = 33;

typedef int name;
typedef string namestring;
typedef unsigned int length;

class names
{

  private:
    vector<namestring> namelist; //Stores a list of reserved and declared names

  public:
    name lookup(namestring str);
    /* Returns the internal representation of the name given in character  */
    /* form.  If the name is not already in the name table, it is          */
    /* automatically inserted.                                             */

    name cvtname(namestring str);
    /* Returns the internal representation of the name given in character  */
    /* form.  If the name is not in the name table then 'blankname' is     */
    /* returned.                                                           */

    void writename(name id);
    /* Prints out the given name on the console                            */

    int namelength(name id);
    /* Returns length ie number of characters in given name               */

    namestring getnamestring(name id);
    /* Returns the namestring for the given name                           */

    names(void);
    /* names initialises the name table.  This procedure is called at      */
    /* system initialisation before any of the above procedures/functions  */
    /* are used.                                                           */
};

#endif /* names_h */
```

Listing 1: names.h

A.1.2 names.cc

```
#include "names.h"
#include <iostream>
#include <string>
#include <cstdlib>

using namespace std;

/* Name storage and retrieval routines */
```

```
9
10 names::names(void)  /* the constructor */
11 {
12   //Populate namelist with reserved words
13   namelist.push_back("DEVICES"); //0
14   namelist.push_back("CONNECTIONS"); //1
15   namelist.push_back("MONITORS"); //2
16   namelist.push_back("END"); //3
17   namelist.push_back("CLOCK"); //4
18   namelist.push_back("SWITCH"); //5
19   namelist.push_back("AND"); //6
20   namelist.push_back("NAND"); //7
21   namelist.push_back("OR"); //8
22   namelist.push_back("NOR"); //9
23   namelist.push_back("DTYPE"); //10
24   namelist.push_back("XOR"); //11
25   namelist.push_back("SIGGEN"); //12
26   namelist.push_back("I1"); //13
27   namelist.push_back("I2"); //14
28   namelist.push_back("I3"); //15
29   namelist.push_back("I4"); //16
30   namelist.push_back("I5"); //17
31   namelist.push_back("I6"); //18
32   namelist.push_back("I7"); //19
33   namelist.push_back("I8"); //20
34   namelist.push_back("I9"); //21
35   namelist.push_back("I10"); //22
36   namelist.push_back("I11"); //23
37   namelist.push_back("I12"); //24
38   namelist.push_back("I13"); //25
39   namelist.push_back("I14"); //26
40   namelist.push_back("I15"); //27
41   namelist.push_back("I16"); //28
42   namelist.push_back("DATA"); //29
43   namelist.push_back("CLK"); //30
44   namelist.push_back("SET"); //31
45   namelist.push_back("CLEAR"); //32
46   namelist.push_back("Q"); //33
47   namelist.push_back("QBAR"); //34
48 }
49
50 name names::lookup(namestring str)
51 {
52   if (cvtname(str) == blankname)
53   {
54     namelist.push_back(str);  //Insert new string
55     return namelist.size() - 1; //Return new strings internal name
56   }
57   else
58   {
59     return cvtname(str);
60   }
61 }
62
63 name names::cvtname(namestring str)
64 {
65   if (str == "") return blankname;
66   for (name id = 0; id < namelist.size(); id++)
67   {
68     if (namelist[id] == str) return id;   //Linear search of namelist vector
69   }
70   return blankname;
71 }
72
73 void names::writename(name id)
74 {
75   if (id == blankname) cout << "blankname";
76   else if (id > blankname && id < namelist.size()) cout << namelist[id];
77   else cout << "Incorrect id";
78 }
79
80 int names::namelength(name id)
81 {
82   if (id > blankname && id < namelist.size()) return namelist[id].length();
83   else return blankname;
84 }
```

```
85
86 namestring names::getnamestring(name id)
87 {
88   if (id > blankname && id < namelist.size()) return namelist[id];
89   else return "";
90 }
```

Listing 2: names.cc

A.2 Scanner Class

A.2.1 scanner.h

```
1  #ifndef scanner_h
2  #define scanner_h
3  #include <string>
4  #include <iostream>
5  #include <fstream>
6  #include <cstdlib>
7  #include "names.h"
8
9  using namespace std;
10
11 typedef int name;
12 typedef enum {namesym, numsym, devsym, consym, monsym, endsym, classsym, iosym, colon, semicol
       , equals, dot, badsym, eofsym} symbol;
13
14 class scanner
15 {
16   public:
17     symbol s;
18     names* nmz;//Pointer to instance of names class
19
20     scanner(names* names_mod,    //Pointer to names class
21         const char* defname,  //Name of file being read
22         bool& ok);          //True of file has been opened correctly
23     ~scanner();             //Destructor
24     void getsymbol(symbol& s,    //Symbol type read
25               name& id,       //Return symbol name (if it has one)
26               int& num,
27               string& numstring);       //Return symbol value (if it's a number)
28     void writelineerror();
29
30   private:
31     ifstream inf; //Input file
32     char curch;   //Current input character
33     char prevch;  //Previous input character. Used for finding line end
34     bool eofile;  //True for end of file
35     bool ok;     //True if the file has been opened correctly
36     int linenum;  //Number of lines in definition file
37     int cursymlen;  //Length of current symbol. Used for error printing
38     string line;  //Current line contents. Used for error printing
39
40     void getch(); //Gets next input character
41     void getnumber(int& number, string& numstring); //Reads number from file
42     void getname(name& id); //Reads name from file
43     string getline(); //Reads the line
44     void skipspaces(); //Skips spaces
45     void skipcomments(); //Skips comments
46 };
47
48 #endif
```

Listing 3: scanner.h

A.2.2 scanner.cc

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

```
3
4  using namespace std;
5
6  scanner::scanner(names* names_mod, const char* defname, bool& ok)
7  {
8    nmz = names_mod;
9    ok = 1;
10   inf.open(defname);  //Open file
11   if (!inf)
12   {
13     cout << "Error: cannot open file for reading" << endl;
14     ok = 0;
15   }
16   eofile = (inf.get(curch) == 0); //Get first character
17   linenum = 1;
18   s = badsym;//in case getline is called before getsymbol
19 }
20
21 scanner::~scanner()
22 {
23   inf.close();  //Close file
24 }
25
26 void scanner::getsymbol(symbol& s, name& id, int& num, string& numstring)
27 {
28   s = badsym;
29   cursymlen = 0;
30   skipspaces();
31   if (eofile) s = eofsym;
32   else
33   {
34     if (isdigit(curch))
35     {
36       s = numsym;
37       getnumber(num, numstring);
38     }
39     else
40     {
41       if (isalpha(curch))
42       {
43         getname(id);
44         if (id == 0) s = devsym;
45         else if (id == 1) s = consym;
46         else if (id == 2) s = monsym;
47         else if (id == 3) s = endsym;
48         else if (id > 3 && id < 13) s = classsym;
49         else if (id > 12 && id < 35) s = iosym;
50         else s = namesym;
51       }
52       else
53       {
54         switch (curch)
55         {
56           case '=':
57             s = equals;
58             getch();
59             break;
60           case ';':
61             s = semicol;
62             getch();
63             break;
64           case ':':
65             s = colon;
66             getch();
67             break;
68           case '.':
69             s = dot;
70             getch();
71             break;
72           case '/':
73             getch();
74             if (curch == '*')
75             {
76               getch();
77               skipcomments();
78               getsymbol(s, id, num, numstring);
```

```
79              }
80              break;
81            default:
82              s = badsym;
83              getch();
84              break;
85          }
86          cursymlen = 1;
87        }
88      }
89    }
90  }
91
92  void scanner::writelineerror()
93  {
94    string errorptr;
95    for (int i = 0; i < ((int)line.length() - cursymlen); i++)
96    {
97      errorptr.push_back(' ');
98    }
99    errorptr.push_back('^');
100   cout << "Line " << linenum << ":" << endl;
101   cout << getline() << endl;     //Outputs current line
102   cout << errorptr << endl; //Outputs a caret at the error
103 }
104
105 void scanner::getch()
106 {
107   prevch = curch;
108   eofile = (inf.get(curch) == 0); //get next character
109   if (prevch == '\n') //If eoline, clear the currently stored line
110   {
111     linenum++;
112     line.clear();
113   }
114   else if (prevch != '\r') //If we're not at the end of a line, add the char to the line
        string
115   {
116     line.push_back(prevch);
117   }
118 }
119
120 void scanner::getnumber(int& number, string& numstring)
121 {
122   numstring = "";
123   number = 0;
124   cursymlen = 0;
125   while (isdigit(curch) && !eofile)
126   {
127     numstring.push_back(curch);
128     number *= 10;
129     number += (int(curch) - int('0'));
130     cursymlen++;
131     getch();
132   }
133 }
134
135 void scanner::getname(name& id)
136 {
137   namestring str;
138   cursymlen = 0;
139   while (isalnum(curch) && !eofile)
140   {
141     str.push_back(curch);
142     cursymlen++;
143     getch();
144   }
145   id = nmz->lookup(str);
146 }
147
148 void scanner::skipspaces()
149 {
150   while (isspace(curch) || curch == '\n')
151   {
152     getch();
153     if (eofile) break;
```

```
154   }
155 }
156
157 void scanner::skipcomments()
158 {
159   while (!(prevch == '*' && curch == '/'))
160   {
161     getch();
162     if (eofile)
163     {
164       cout << "Reached end of file before comment was terminated" << endl;
165       break;
166     }
167   }
168   getch(); //Get to next useful char
169 }
170
171 string scanner::getline()
172 {
173   if (s != semicol)
174   {
175     while (curch != ';' && !eofile && curch != '\n')
176     {
177       getch();
178     }
179     if (curch != '\n' && curch != '\r' && !eofile)
180     {
181       line.push_back(curch);
182     }
183   }
184   return line;
185 }
```

Listing 4: scanner.cc

A.3 Parser Class

A.3.1 parser.cc

```
1  #include <iostream>
2  #include "parser.h"
3  #include "error.h"
4
5  using namespace std;
6
7  /* The parser for the circuit definition files */
8
9  bool parser::readin(void)
10 {
11   //EBNF: specfile = devices connections monitors
12   bool deviceDone = false, connectionDone = false, monitorDone = false;
13   devicePresent = connectionPresent = monitorPresent = false;
14   cursym = badsym;
15   while (cursym != eofsym)
16   {
17     if (cursym != devsym && cursym != consym && cursym != monsym)
18     {
19       smz->getsymbol(cursym, curname, curint, numstring);
20     }
21     if (cursym == devsym)
22     {
23       if (deviceDone)
24       {
25         erz->newError(25);//Must only be one devices list
26       }
27       deviceDone = true;
28       deviceList();
29     }
30     else if (cursym == consym)
31     {
32       if (!deviceDone)
33       {
```

```
34          erz->newError(0); //must have device list first
35        }
36        if (connectionDone)
37        {
38          erz->newError(28);//Must only be one connections list
39
40        }
41        connectionDone = true;
42        connectionList();
43      }
44      else if (cursym == monsym)
45      {
46        if (!deviceDone | !connectionDone)
47        {
48          erz->newError(2); //Must have monitor list last
49        }
50        if (monitorDone)
51        {
52          erz->newError(29);//Must only be one Monitors list
53        }
54        monitorDone = true;
55        monitorList();
56      }
57      else if (cursym != eofsym)
58      {
59        while (cursym != devsym && cursym != consym && cursym != monsym && cursym != eofsym)
60        {
61          smz->getsymbol(cursym, curname, curint, numstring);
62          erz->countSymbols();
63        }
64        erz->symbolError(deviceDone, connectionDone, monitorDone);
65      }
66    }
67    if (!deviceDone)
68    {
69      erz->newError(26);//There must be a DEVICES block, it may not have been initialised
      properly
70    }
71    if (!connectionDone)
72    {
73      erz->newError(30);//There must be a CONNECTIONS block, it may not have been initialised
      properly
74    }
75    if (!monitorDone)
76    {
77      erz->newError(31);//There must be a MONITORS block, it may not have been initialised
      properly
78    }
79    netz->checknetwork(correctOperation);
80    anyErrors = erz->anyErrors();
81    return (correctOperation && !anyErrors);
82  }
83
84  void parser::deviceList()
85  {
86    //EBNF: devices = 'DEVICES' dev {';' dev} ';' 'END'
87    bool deviceError;
88    if (!devicePresent)
89    {
90      smz->getsymbol(cursym, curname, curint, numstring);
91      if (cursym == classsym)
92      {
93        deviceError = newDevice(curname);
94        devicePresent = true;
95      }
96      else if (cursym == endsym)
97      {
98        erz->newError(3); //must have at least one device
99        return;
100     }
101     else
102     {
103       erz->newError(4); //need a device type
104     }
105     if (!deviceError)
106     {
```

```
107         smz->getsymbol(cursym, curname, curint, numstring);
108       }
109     }
110     while (cursym == semicol)
111     {
112       smz->getsymbol(cursym, curname, curint, numstring);
113       if (cursym == classsym)
114       {
115         deviceError = newDevice(curname);
116       }
117       else if (cursym == endsym)
118       {
119         return;
120       }
121       else if (cursym == consym | cursym == devsym | cursym == monsym)
122       {
123         erz->newError(32);//Block must be terminated with 'END'
124         return;
125       }
126       else
127       {
128         erz->newError(5);//Expecting device name or END after semicolon (device name must start
      with letter)
129       }
130       if (!deviceError)
131       {
132         smz->getsymbol(cursym, curname, curint, numstring);
133       }
134     }
135     if (!deviceError) erz->newError(24);//must end line in semicolon
136     while (cursym != semicol && cursym != endsym && cursym != eofsym)
137     {
138       smz->getsymbol(cursym, curname, curint, numstring);
139     }
140     if (cursym == semicol)
141     {
142       deviceList();
143     }
144     if (cursym == endsym)
145     {
146       return;
147     }
148 }
149
150
151 bool parser::newDevice(int deviceType)
152 {
153   //EBNF: dev = clock|switch|gate|dtype|xor|siggen
154   bool errorOccurance = false;
155   smz->getsymbol(cursym, curname, curint, numstring);
156   if (cursym == namesym)
157   {
158     devlink nameCheck = netz->finddevice(curname);
159     if(nameCheck==NULL)
160     {
161       name devName = curname;
162       if (deviceType == 10)
163       {
164         dmz->makedevice(dtype, devName, 0, correctOperation); //create DTYPE with name devName
165         return errorOccurance;
166       }
167       if (deviceType == 11)
168       {
169         dmz->makedevice(xorgate, devName, 2, correctOperation); //create XOR with name devName
170         return errorOccurance;
171       }
172       smz->getsymbol(cursym, curname, curint, numstring);
173       if (cursym == colon)
174       {
175         smz->getsymbol(cursym, curname, curint, numstring);
176         if (cursym == numsym)
177         {
178           switch (deviceType)
179           {
180             case 4:
181               if (curint > 0)
```

```
182                 {
183                   dmz->makedevice(aclock, devName, curint, correctOperation); //create clock
    with curint and devName
184                 }
185                 else
186                 {
187                   erz->newError(6);//clock must have number greater than 0
188                   errorOccurance=true;
189                 }
190                 break;
191               case 5:
192                 if (curint == 1 || curint == 0)
193                 {
194                   dmz->makedevice(aswitch, devName, curint, correctOperation);//create switch
    with curint and devName
195                 }
196                 else
197                 {
198                   erz->newError(7);//switch must have either 0 or 1
199                   errorOccurance=true;
200                 }
201                 break;
202               case 6:
203               case 7:
204               case 8:
205               case 9:
206                 if (curint > 0 && curint < 17)
207                 {
208                   switch (deviceType)
209                   {
210                     case 6:
211                       dmz->makedevice(andgate, devName, curint, correctOperation);//create and
    gate with curint and devName
212                       break;
213                     case 7:
214                       dmz->makedevice(nandgate, devName, curint, correctOperation);//create nand
     gate with curint and devName
215                       break;
216                     case 8:
217                       dmz->makedevice(orgate, devName, curint, correctOperation);//create or
    gate with curint and devName
218                       break;
219                     case 9:
220                       dmz->makedevice(norgate, devName, curint, correctOperation);//create nor
    gate with curint and devName
221                       break;
222                     default:
223                       cout << "How on earth have you managed to get here?" << endl;
224                   }
225                 }
226                 else
227                 {
228                   erz->newError(8);//must have between 1 and 16 inputs to a GATE
229                   errorOccurance=true;
230                 }
231                 break;
232               case 12:
233                 if (isBinary(numstring))
234                 {
235                   sequence waveform;
236                   for (int i=0; i<numstring.length(); i++)
237                   {
238                     waveform.push_back(numstring[i]=='1');
239                   }
240                   dmz->makesiggen(devName, waveform); //create SIGGEN with name devName
241                 }
242                 else
243                 {
244                   erz->newError(36); //Must be a binary input
245                   errorOccurance=true;
246                 }
247                 break;
248               default:
249                 cout << "Please do not deduct marks if this message is displayed" << endl;
250             }
251             return errorOccurance;
```

```
252           }
253           else
254           {
255             erz->newError(9);//clock needs clock cycle number
256             errorOccurance=true;
257           }
258         }
259         else
260         {
261           erz->newError(10);//need colon after name for CLOCK/SWITCH/GATE type
262           errorOccurance=true;
263         }
264       }
265       else
266       {
267         erz->newError(27);//attempting to give two devices the same name, choose an alternative
      name
268         errorOccurance=true;
269       }
270     }
271     else if (cursym!=badsym)
272     {
273       erz->newError(33);//using reserved word as device name
274       errorOccurance=true;
275     }
276     else
277     {
278       erz->newError(11);//name must begin with letter and only containing letter number and _
279       errorOccurance=true;
280     }
281     return errorOccurance;
282   }
283
284   void parser::connectionList()
285   {
286     //EBNF: connections = 'CONNECTIONS' {con ';'} 'END'
287     bool connectionError;
288     if (!connectionPresent)
289     {
290       smz->getsymbol(cursym, curname, curint, numstring);
291       if (cursym == endsym)
292       {
293         if (!connectionPresent)
294         {
295           erz->newWarning(0);//No Connections
296         }
297         return;
298       }
299       else if (cursym == namesym)
300       {
301         connectionError = newConnection();
302         connectionPresent = true;
303       }
304       else
305       {
306         erz->newError(12);//connection must start with the name of a device
307       }
308       if (!connectionError)
309       {
310         smz->getsymbol(cursym, curname, curint, numstring);
311       }
312     }
313     while (cursym == semicol)
314     {
315       smz->getsymbol(cursym, curname, curint, numstring);
316       if (cursym == namesym)
317       {
318         connectionError = newConnection();
319       }
320       else if (cursym == endsym)
321       {
322         return;
323       }
324       else if (cursym == consym | cursym == devsym | cursym == monsym)
325       {
326         erz->newError(32);//Block must be terminated with 'END'
```

```
327         return;
328       }
329       else
330       {
331         erz->newError(13);//connection must start with the name of a device or end of device
      list must be terminated with END (not semicolon)
332       }
333       if (!connectionError)
334       {
335         smz->getsymbol(cursym, curname, curint, numstring);
336       }
337     }
338     if (!connectionError) erz->newError(24);//must end line in semicolon
339     while (cursym != semicol && cursym != endsym && cursym != eofsym)
340     {
341       smz->getsymbol(cursym, curname, curint, numstring);
342     }
343     if (cursym == semicol)
344     {
345       connectionList();
346     }
347     if (cursym == endsym)
348     {
349       return;
350     }
351   }
352
353   bool parser::newConnection()
354   {
355     //EBNF: con = devicename'.'input '=' devicename['.'output]
356     bool errorOccurance = false;
357     devlink devtype = netz->finddevice(curname);
358     if (devtype != NULL)
359     {
360       connectionInName = curname;
361       smz->getsymbol(cursym, curname, curint, numstring);
362       if (cursym == dot)
363       {
364         smz->getsymbol(cursym, curname, curint, numstring);
365         devtype = netz->finddevice(connectionInName);
366         inplink ilist = netz->findinput(devtype, curname);
367         if (cursym == iosym && ilist != NULL)
368         {
369           name inputPin = curname;
370           smz->getsymbol(cursym, curname, curint, numstring);
371           if (cursym == equals) //SEARCH - you have got to here
372           {
373             smz->getsymbol(cursym, curname, curint, numstring);
374             devtype = netz->finddevice(curname);
375             if (devtype != NULL)
376             {
377               connectionOutName = curname;
378               switch (devtype ? devtype->kind : baddevice)
379               {
380                 case 7:
381                   smz->getsymbol(cursym, curname, curint, numstring);
382                   if (cursym == dot)
383                   {
384                     smz->getsymbol(cursym, curname, curint, numstring);
385                     outplink olist = netz->findoutput(devtype, curname);
386                     if (cursym == iosym && olist != NULL)
387                     {
388                       netz->makeconnection(connectionInName, inputPin, connectionOutName,
      curname, correctOperation);
389                       return errorOccurance;
390                     }
391                     else
392                     {
393                       erz->newError(34); //Not valid output for dtype
394                     }
395                   }
396                   else
397                   {
398                     erz->newError(14);  //Expect a dot after dtype
399                     errorOccurance=true;
400                   }
```

```
401                 default:
402                 //check the connection is unique
403                   if (ilist->connect==NULL)
404                   {
405                     netz->makeconnection(connectionInName, inputPin, connectionOutName,
    blankname, correctOperation);
406                     return errorOccurance;
407                   }
408                   else if (ilist->connect==netz->findoutput(devtype, blankname))
409                   {
410                     namestring repeatedInput = smz->nmz->getnamestring(connectionInName);
411                     namestring repeatedOutput = smz->nmz->getnamestring(connectionOutName);
412                     erz->connectionWarning(repeatedInput, repeatedOutput);
413                   }
414                   else
415                   {
416                     erz->newError(37);//attempting to input 2 ouputs into same input
417                   }
418               }
419             }
420             else
421             {
422               erz->newError(15); //Device does not exist
423               errorOccurance=true;
424             }
425           }
426           else
427           {
428             erz->newError(16);//Must specify output to connect to input with equals sign
429             errorOccurance=true;
430           }
431         }
432         else
433         {
434           erz->newError(17);//specify valid input gate after dot
435           errorOccurance=true;
436         }
437       }
438       else
439       {
440         erz->newError(18);//need to seperate connection input with a '.' (or need to specify
    input)
441         errorOccurance=true;
442       }
443     }
444     else
445     {
446       erz->newError(19); //Device does not exist
447       errorOccurance=true;
448     }
449     return errorOccurance;
450 }
451
452 void parser::monitorList()
453 {
454   //EBNF: monitors = 'MONITORS' {mon ';'} 'END'
455   bool monitorError;
456   if (!monitorPresent)
457   {
458     smz->getsymbol(cursym, curname, curint, numstring);
459     if (cursym == endsym)
460     {
461       if (!monitorPresent)
462       {
463         erz->newWarning(1);//No Monitors
464       }
465       return;
466     }
467     else if (cursym == namesym)
468     {
469       monitorError = newMonitor();
470       monitorPresent = true;
471     }
472     else
473     {
474       erz->newError(20);//monitor must start with the name of a device
```

```
475       }
476       if (!monitorError)
477       {
478         smz->getsymbol(cursym, curname, curint, numstring);
479       }
480     }
481     while (cursym == semicol)
482     {
483       smz->getsymbol(cursym, curname, curint, numstring);
484       if (cursym == namesym)
485       {
486         monitorError = newMonitor();
487       }
488       else if (cursym == endsym)
489       {
490         return;
491       }
492       else if (cursym == consym | cursym == devsym | cursym == monsym)
493       {
494         erz->newError(32);//Block must be terminated with 'END'
495         return;
496       }
497       else
498       {
499         erz->newError(21);//monitor must start with the name of a device or end of device list must be terminated with END (not semicolon)
500       }
501       if (!monitorError)
502       {
503         smz->getsymbol(cursym, curname, curint, numstring);
504       }
505     }
506     if (!monitorError) erz->newError(24);//must end line in semicolon
507     while (cursym != semicol && cursym != endsym && cursym != eofsym)
508     {
509       smz->getsymbol(cursym, curname, curint, numstring);
510     }
511     if (cursym == semicol)
512     {
513       monitorList();
514     }
515     if (cursym == endsym)
516     {
517       return;
518     }
519   }
520
521   bool parser::newMonitor()
522   {
523     //EBNF: mon = devicename['.'output]
524     bool errorOccurance = false;
525     devlink devtype = netz->finddevice(curname);
526     if (devtype != NULL)
527     {
528       monitorName = curname;
529       switch (devtype ? devtype->kind : baddevice)
530       {
531         case 7:
532           smz->getsymbol(cursym, curname, curint, numstring);
533           if (cursym == dot)
534           {
535             smz->getsymbol(cursym, curname, curint, numstring);
536             outplink olist = netz->findoutput(devtype, curname);
537             bool alreadyMonitored = mmz->IsMonitored(olist);
538             if (!alreadyMonitored)
539             {
540               if (cursym == iosym && olist != NULL)
541               {
542                 mmz->makemonitor(monitorName, curname, correctOperation);
543                 return errorOccurance;
544               }
545               else
546               {
547                 erz->newError(34); //Not valid output for dtype
548               }
549             }
```

```
550             else
551             {
552               namestring repeatedMonitor = smz->nmz->getnamestring(curname);
553               erz->monitorWarning(repeatedMonitor); //repeated monitors
554               if (cursym == iosym && olist != NULL)
555               {
556                 mmz->makemonitor(monitorName, curname, correctOperation);
557                 return errorOccurance;
558               }
559               else
560               {
561                 erz->newError(34); //Not valid output for dtype
562               }
563             }
564           }
565           else
566           {
567             erz->newError(22);  //Expect a dot after dtype
568             errorOccurance=true;
569           }
570         default:
571           outplink olist = netz->findoutput(devtype, blankname);
572           bool alreadyMonitored = mmz->IsMonitored(olist);
573             if (!alreadyMonitored)
574             {
575               mmz->makemonitor(monitorName, blankname, correctOperation);
576             }
577             else
578             {
579               namestring repeatedMonitor = smz->nmz->getnamestring(curname);
580               erz->monitorWarning(repeatedMonitor); //repeated monitors
581               mmz->makemonitor(monitorName, curname, correctOperation);
582             }
583           return errorOccurance;
584       }
585     }
586     else
587     {
588       erz->newError(23);//bad device monitor
589       errorOccurance=true;
590     }
591     return errorOccurance;
592 }
593
594 bool parser::isBinary(string numstring)
595 {
596   for (int i=0; i<numstring.length(); i++)
597   {
598     if (numstring[i]!='0' && numstring[i]!='1')
599       {
600         return false;
601       }
602   }
603   return true;
604 }
605
606 parser::parser(network* network_mod, devices* devices_mod, monitor* monitor_mod, scanner*
      scanner_mod, error* error_mod)
607 {
608   netz = network_mod;  /* make internal copies of these class pointers */
609   dmz = devices_mod;   /* so we can call functions from these classes  */
610   mmz = monitor_mod;   /* eg. to call makeconnection from the network  */
611   smz = scanner_mod;   /* class you say:                               */
612   erz = error_mod; /* netz->makeconnection(i1, i2, o1, o2, ok);   */
613   /* any other initialisation you want to do? */
614 }
```

Listing 5: parser.cc

`parser.cc` was written with joint effort between myself and Tim Hillel, with Tim contributing approximately 75% of the code.

A.4 Test Scripts

A.4.1 test.sh

```
#!/bin/bash
echo "" > error.txt
for f in *.gf2
do
  echo "processing $f..."
  echo "processing $f..." >> test.txt
  echo -e "r 50\r\nq\r\n" | ../src/logsim $f >> test.txt
  echo -e '\n' >> test.txt
done
```

Listing 6: test.sh

A.4.2 error.sh

```
#!/bin/bash
echo "" > error.txt
for f in *.gf2
do
  echo "processing $f..."
  echo "processing $f..." >> error.txt
  echo ../../src/logsim $f >> error.txt
  echo -e '\n' >> error.txt
done
```

Listing 7: error.sh

B Test Definition Files

All supplied definition files and circuit diagrams were written and designed by myself.

B.1 XOR Gate

B.1.1 Definition File

```
DEVICES
SWITCH S1:0;
SWITCH S2:1;
NAND G1:2;
NAND G2:2;
NAND G3:2;
NAND G4:2;
END

CONNECTIONS
G1.I1 = S1;
G1.I2 = S2;
G2.I1 = S1;
G2.I2 = G1;
G3.I1 = G1;
G3.I2 = S2;
G4.I1 = G2;
G4.I2 = G3;
END

MONITORS
S1;
S2;
G4;
END
```

Listing 8: xor.gf2

B.1.2 Circuit Diagram

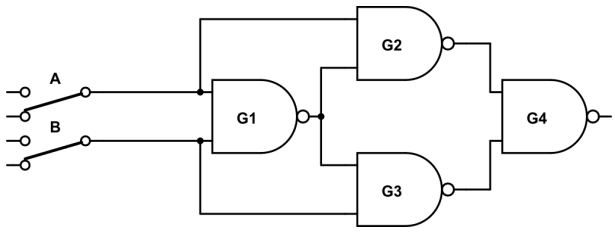

Figure 2: Circuit diagram of an XOR gate implemented using NAND gates

B.2 4-bit Adder

B.2.1 Definition File

```
DEVICES
/* 4 bit inputs */
SWITCH A0:1;
SWITCH A1:0;
SWITCH A2:0;
SWITCH A3:0;
SWITCH B0:1;
SWITCH B1:0;
SWITCH B2:0;
SWITCH B3:0;
```

```
SWITCH C0:1; /* Carry in */
AND AND1:2;
AND AND2:2;
AND AND3:2;
AND AND4:2;
AND AND5:2;
AND AND6:2;
AND AND7:2;
AND AND8:2;
XOR XOR1;
XOR XOR2;
XOR XOR3;
XOR XOR4;
XOR XOR5;
XOR XOR6;
XOR XOR7;
XOR XOR8;
OR OR1:2;
OR OR2:2;
OR OR3:2;
OR OR4:2;
END

CONNECTIONS
/* LSB adder */
XOR1.I1 = A0;
XOR1.I2 = B0;
AND1.I1 = XOR1;
AND1.I2 = C0;
AND2.I1 = A0;
AND2.I2 = B0;
XOR2.I1 = XOR1;
XOR2.I2 = C0;
OR1.I1 = AND1;
OR1.I2 = AND2;

XOR3.I1 = A1;
XOR3.I2 = B1;
AND3.I1 = XOR3;
AND3.I2 = OR1;
AND4.I1 = A1;
AND4.I2 = B1;
XOR4.I1 = XOR3;
XOR4.I2 = OR1;
OR2.I1 = AND3;
OR2.I2 = AND4;

XOR5.I1 = A2;
XOR5.I2 = B2;
AND5.I1 = XOR5;
AND5.I2 = OR2;
AND6.I1 = A2;
AND6.I2 = B2;
XOR6.I1 = XOR5;
XOR6.I2 = OR2;
OR3.I1 = AND5;
OR3.I2 = AND6;

/* MSB Adder */
XOR7.I1 = A3;
XOR7.I2 = B3;
AND7.I1 = XOR7;
AND7.I2 = OR3;
AND8.I1 = A3;
AND8.I2 = B3;
XOR8.I1 = XOR7;
XOR8.I2 = OR3;
OR4.I1 = AND7;
OR4.I2 = AND8;
END

MONITORS
/* Outputs */
XOR2;
XOR4;
XOR6;
```

```
XOR8;
OR4; /* Carry out */
END
```

Listing 9: 4bitadder.gf2

B.2.2 Circuit Diagram

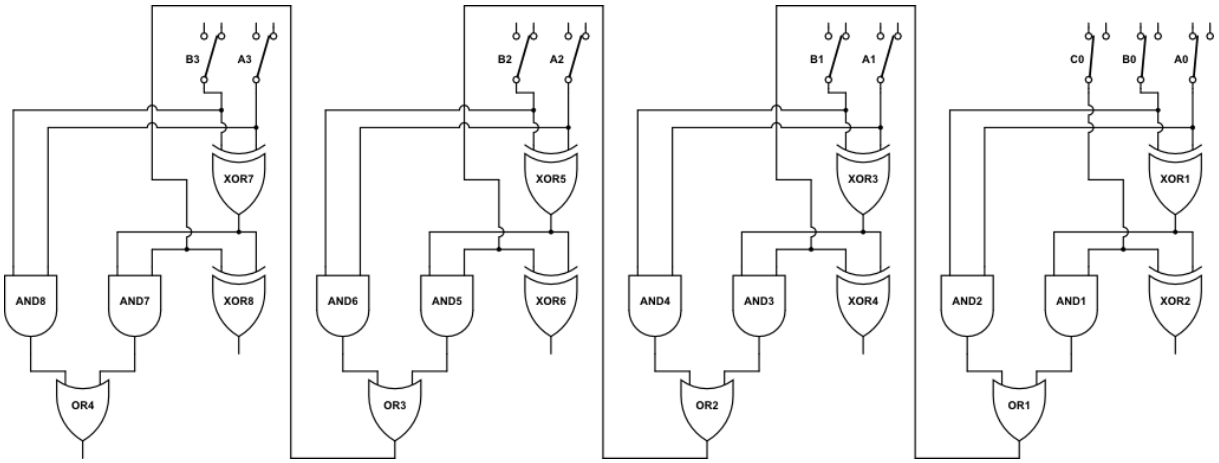

Figure 3: Circuit diagram of a 4-bit adder

B.3 Serial In Parallel Out Shift Register

B.3.1 Definition File

```
DEVICES
CLOCK CLK1:2;
CLOCK CLK2:1;
SWITCH S:0; /* Set switch */
SWITCH R:0; /* Reset switch */
DTYPE D1;
DTYPE D2;
DTYPE D3;
DTYPE D4;
END

CONNECTIONS
D1.DATA = CLK1;
D2.DATA = D1.Q;
D3.DATA = D2.Q;
D4.DATA = D3.Q;
D1.CLK = CLK2;
D2.CLK = CLK2;
D3.CLK = CLK2;
D4.CLK = CLK2;
D1.SET = S;
D2.SET = S;
D3.SET = S;
D4.SET = S;
D1.CLEAR = R;
D2.CLEAR = R;
D3.CLEAR = R;
D4.CLEAR = R;
END

MONITORS
CLK2;
D1.Q;
D2.Q;
D3.Q;
D4.Q;
```

```
END
```

Listing 10: sipo.gf2

B.3.2 Circuit Diagram

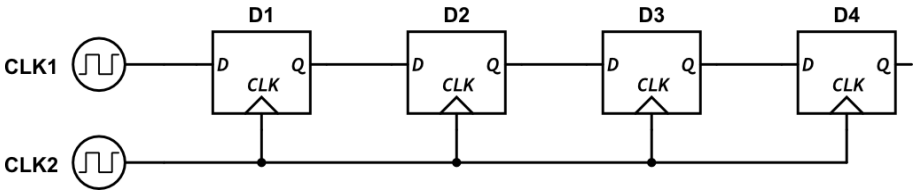

Figure 4: Circuit diagram of a serial in parallel out shift register

NB The software used to draw the circuit diagram does not support the same style of D flip-flop used in the definition file, and Fig. 4 was the closest achievable.

B.4 Gated D Latch

B.4.1 Definition File

```
DEVICES
CLOCK CLK1:1;
CLOCK CLK2:2;
NAND G1:1;
AND G2:2;
AND G3:2;
NOR G4:2;
NOR G5:2;
END

CONNECTIONS
G1.I1 = CLK1;
G2.I1 = G1;
G2.I2 = CLK2;
G3.I1 = CLK2;
G3.I2 = CLK1;
G4.I1 = G2;
G4.I2 = G5;
G5.I1 = G4;
G5.I2 = G3;
END

MONITORS
CLK1; /* D */
CLK2; /* E */
G4; /* Q */
G5; /* QBAR */
END
```

Listing 11: sipo.gf2

B.4.2 Circuit Diagram

NB The software used to draw the circuit diagram does not support the NAND gates with one input. Therefore the NAND gate G1 was substituted for a NOT gate as can be seen in Fig. 5.

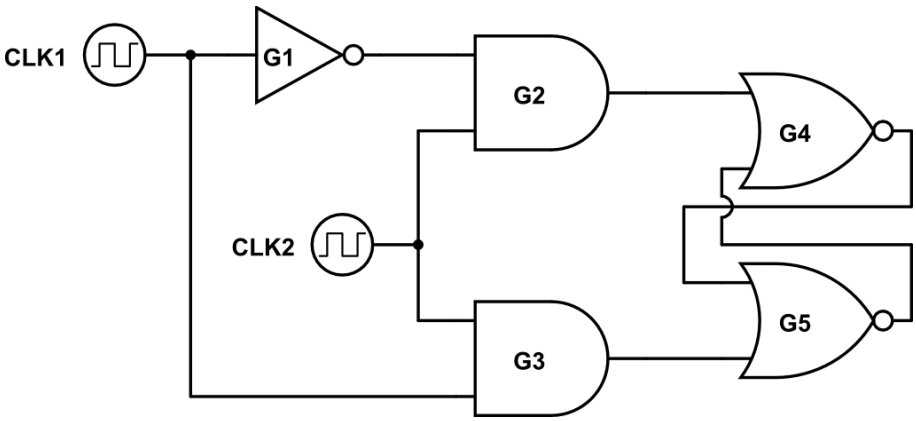

Figure 5: Circuit diagram of a Gated D Latch

C EBNF

```
specfile = devices connections monitors

devices = 'DEVICES' dev ';' {dev ';'} 'END'
connections = 'CONNECTIONS' {con ';'} 'END'
monitors = 'MONITORS' {mon ';'} 'END'

dev = clock|switch|gate|dtype|xor|siggen
con = devicename'.'input '=' devicename['.'output]
mon = devicename['.'output]

devicename = letter {'_'|letter|digit}
input = 'I'('1'|'2'|'3'|'4'|'5'|'6'|'7'|'8'|'9'|
  '10'|'11'|'12'|'13'|'14'|'15'|'16')|'DATA'|'CLK'|'SET'|'CLEAR'
output = 'Q'['BAR']

clock = 'CLOCK' devicename':'digit{digit}
switch = 'SWITCH' devicename':'('0'|'1')
gate = ('AND'|'NAND'|'OR'|'NOR') devicename':'('1'|'2'|'3'|'4'|'5'|'6'|'7'|'8'|
  '9'|'10'|'11'|'12'|'13'|'14'|'15'|'16')
dtype = 'DTYPE' devicename
xor = 'XOR' devicename
siggen = 'SIGGEN' devicename':'('0'|'1'){'0'|'1'}
```

Listing 12: EBNF

D User Guide

E File Listing