

IIA GF2 Software: Final Report

Jamie Magee (jam96)
Team 8
Gonville & Caius College

Contents

1	Description	2
2	Development Style	2
3	My Contribution	3
4	Testing	3
5	Conclusions	3
A	Code Listings	5
A.1	Names Class	5
A.1.1	names.h	5
A.1.2	names.cc	5
A.2	Scanner Class	7
A.2.1	scanner.h	7
A.2.2	scanner.cc	7
A.3	Parser Class	10
A.3.1	parser.cc	10
A.4	Test Scripts	19
A.4.1	test.sh	19
A.4.2	error.sh	19
B	Test Definition Files	20
B.1	XOR Gate	20
B.1.1	Definition File	20
B.1.2	Circuit Diagram	20
B.2	4-bit Adder	20
B.2.1	Definition File	20
B.2.2	Circuit Diagram	22
B.3	Serial In Parallel Out Shift Register	22
B.3.1	Definition File	22
B.3.2	Circuit Diagram	23
B.4	Gated D Latch	23
B.4.1	Definition File	23
B.4.2	Circuit Diagram	23
B.5	Signal Generator Example	24
B.5.1	Definition File	24
B.5.2	Circuit Diagram	24
C	EBNF	25
D	User Guide	26
E	File Listing	27
F	Gantt Chart	28

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

It uses files written in the language as specified in our EBNF in Appendix C, with the file extension `.gf2` and have MIME type `text/plain`. Errors in a definition file are reported fully to the user upon reading a definition file, and give unique error codes for each different type of error. There is no limit on the number of devices in a network (except for that implied by the available memory of the computer). There may also be an unlimited number of monitors. For a complete guide on how to operate our logic simulator, please see the user guide in Appendix D.

For the structure of our logic simulator we made use of modularisation through the use of object oriented programming. The logic circuit is internally represented using the network class, in addition to the devices class which stores details about the properties of each device. When a file is opened, it is read by the parser class, by creating an instance of the scanner class

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. A Gantt chart showing our time planning can be seen in Appendix F.

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. It also allowed us to work on the same file independently and merge our changes using the `git merge` command.

During the maintenance phase of the project we divided up the tasks and tackled them independently. They

were assigned as follows:

Non-zero Hold Time of Bistables: J Magee

Signal Generators: T Hillel

Continuous Simulation: M Jackson

While the tasks were split up, and the person assigned to the task contributed the majority of the code, the power of `git` allowed other members of the team to contribute easily.

Overall I believe our team worked efficiently, and as a result we were able to achieve all the requirements of the client within the deadlines we were set.

3 My Contribution

I took on responsibility for writing the names and scanner classes. In addition, I wrote approximately 25% of the parser class. the names class stores a list of all the words used within a definition file, and methods to manipulate them. It is initialised with only the reserved words, but can be populated as a definition file is read. the scanner class reads through the definition file, character by character, and is able to return complete symbols to the parser. It is able to return the internal representation of a symbol, the type of symbol and optionally the value. The parser class analyses the definition file as it is read in according to the rules laid out in our EBNF. It is then able to create devices, connections and monitors that are laid out in the definition file. I wrote the `newConnection`, `monitorList` and `newMonitor` methods. The `newConnection` and `newMonitor` methods deal with reading in a connection and monitor respectively, and creating them using the network and monitor classes respectively. The `monitorList` class deals with reading in an entire `MONITOR` block, as defined by our EBNF, which can be seen in Appendix C

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. In addition to testing for errors, I wrote a shell script which would run through each of our test definition files, in Appendix B, and record the output. Both the shell scripts can be found in Appendix A.4.

During the maintenance phase of the project, I undertook the task to implement a non-zero hold time for d-type flip flops, as well as randomising the order of execution of devices. the non-deterministic characteristics of d-types required editing of the devices class, specifically the `execdtype` method. Randomising the order of execution of devices was implemented in the network class, and required reading the values from the existing devices list, shuffling the list (while still retaining clocks at the end of the list) and writing it back.

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 variations 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

Overall I believe that we worked well as a team, and were able to achieve the requirements set out by the client.

If we had more time I would improve our logic simulator by:

- Implementing a WYSIWYG GUI
- Improving the parser to allow simple error correction
- Expand the device library to include JK flip-flops

A Code Listings

A.1 Names Class

A.1.1 names.h

```
1 #ifndef names_h
2 #define names_h
3
4 #include <string>
5 #include <vector>
6
7 using namespace std;
8
9 //const int maxnames = 200; /* max number of distinct names */
10 //const int maxlength = 8; /* max chars in a name string */
11 const int blankname = -1; /* special name */
12
13 const int lastreservedname = 34;
14
15 typedef int name;
16 typedef string namestring;
17 typedef unsigned int length;
18
19 class names
20 {
21
22 private:
23     vector<namestring> namelist; //Stores a list of reserved and declared names
24
25 public:
26     name lookup(namestring str);
27     /* Returns the internal representation of the name given in character */
28     /* form. If the name is not already in the name table, it is */
29     /* automatically inserted. */
30
31     name cvtname(namestring str);
32     /* Returns the internal representation of the name given in character */
33     /* form. If the name is not in the name table then 'blankname' is */
34     /* returned. */
35
36     void writename(name id);
37     /* Prints out the given name on the console */
38
39     int namelength(name id);
40     /* Returns length ie number of characters in given name */
41
42     namestring getnamestring(name id);
43     /* Returns the namestring for the given name */
44
45     names(void);
46     /* names initialises the name table. This procedure is called at */
47     /* system initialisation before any of the above procedures/functions */
48     /* are used. */
49 };
50
51 #endif /* names_h */
```

Listing 1: names.h

A.1.2 names.cc

```
1 #include "names.h"
2 #include <iostream>
3 #include <string>
4 #include <cstdlib>
5
6 using namespace std;
7
8 /* 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     skipspace();
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 < lastreservedname+1) 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
115         string
116     {
117         line.push_back(prevch);
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
70     properly
71 }
72 if (!connectionDone)
73 {
74     erz->newError(30); //There must be a CONNECTIONS block, it may not have been initialised
75     properly
76 }
77 if (!monitorDone)
78 {
79     erz->newError(31); //There must be a MONITORS block, it may not have been initialised
80     properly
81 }
82 netz->checknetwork(correctOperation);
83 anyErrors = erz->anyErrors();
84 return (correctOperation && !anyErrors);
85 }
86
87 void parser::deviceList()
88 {
89     //EBNF: devices = 'DEVICES' dev { ';' dev } ';' 'END'
90     bool alreadyGotNextSymbol;
91     if (!devicePresent)
92     {
93         smz->getsymbol(cursym, curname, curint, numstring);
94         if (cursym == classsym)
95         {
96             alreadyGotNextSymbol = newDevice(curname);
97             devicePresent = true;
98         }
99     }
100     else if (cursym == endsym)
101     {
102         erz->newError(3); //must have at least one device
103         return;
104     }
105     else
106     {
107         erz->newError(4); //need a device type
108     }
109     if (!alreadyGotNextSymbol)
110     {

```

```

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         alreadyGotNextSymbol = 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
129         with letter)
130     }
131     if (!alreadyGotNextSymbol)
132     {
133         smz->getsymbol(cursym, curname, curint, numstring);
134     }
135 }
136 if (!alreadyGotNextSymbol) erz->newError(24); //must end line in semicolon
137 while (cursym != semicol && cursym != endsym && cursym != eofsym)
138 {
139     smz->getsymbol(cursym, curname, curint, numstring);
140 }
141 if (cursym == semicol)
142 {
143     deviceList();
144 }
145 if (cursym == endsym)
146 {
147     return;
148 }
149 }
150
151 bool parser::newDevice(int deviceType)
152 {
153     //EBNF: dev = clock|switch|gate|dtype|xor|siggen
154     bool symbolSkip = 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 symbolSkip;
166             }
167             if (deviceType == 11)
168             {
169                 dmz->makedevice(xorgate, devName, 2, correctOperation); //create XOR with name devName
170                 return symbolSkip;
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             symbolSkip=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             symbolSkip=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                     break;
224             }
225         }
226         else
227         {
228             erz->newError(8); //must have between 1 and 16 inputs to a GATE
229             symbolSkip=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             smz->getsymbol(cursym, curname, curint, numstring);
241             symbolSkip=true;
242             while (cursym==numsym)
243             {
244                 if (isBinary(numstring))
245                 {
246                     for (int i=0; i<numstring.length(); i++)
247                     {
248                         waveform.push_back(numstring[i]=='1');
249                     }
250                     smz->getsymbol(cursym, curname, curint, numstring);
251                 }

```

```

252         else
253         {
254             erz->newError(36); //Must be a binary input
255             symbolSkip=true;
256             break;
257         }
258     }
259     dmz->makesiggen(devName, waveform); //create SIGGEN with name devName
260 }
261 else
262 {
263     erz->newError(36); //Must be a binary input
264     symbolSkip=true;
265 }
266 break;
267 default:
268     break;
269 }
270 return symbolSkip;
271 }
272 else
273 {
274     erz->newError(9); //clock needs clock cycle number
275     symbolSkip=true;
276 }
277 }
278 else
279 {
280     erz->newError(10); //need colon after name for CLOCK/SWITCH/GATE type
281     symbolSkip=true;
282 }
283 }
284 else
285 {
286     erz->newError(27); //attempting to give two devices the same name, choose an alternative
name
287     symbolSkip=true;
288 }
289 }
290 else if (cursym!=badsym)
291 {
292     erz->newError(33); //using reserved word as device name
293     symbolSkip=true;
294 }
295 else
296 {
297     erz->newError(11); //name must begin with letter and only containing letter number and _
298     symbolSkip=true;
299 }
300 return symbolSkip;
301 }
302
303 void parser::connectionList()
304 {
305     //EBNF: connections = 'CONNECTIONS' {con ';' } 'END'
306     bool connectionError;
307     if (!connectionPresent)
308     {
309         smz->getsymbol(cursym, curname, curint, numstring);
310         if (cursym == endsym)
311         {
312             if (!connectionPresent)
313             {
314                 erz->newWarning(0); //No Connections
315             }
316             return;
317         }
318         else if (cursym == namesym)
319         {
320             connectionError = newConnection();
321             connectionPresent = true;
322         }
323         else
324         {
325             erz->newError(12); //connection must start with the name of a device
326         }

```

```

327     if (!connectionError)
328     {
329         smz->getsymbol(cursym, curname, curint, numstring);
330     }
331 }
332 while (cursym == semicol)
333 {
334     smz->getsymbol(cursym, curname, curint, numstring);
335     if (cursym == namesym)
336     {
337         connectionError = newConnection();
338     }
339     else if (cursym == endsym)
340     {
341         return;
342     }
343     else if (cursym == consym | cursym == devsym | cursym == monsym)
344     {
345         erz->newError(32); //Block must be terminated with 'END'
346         return;
347     }
348     else
349     {
350         erz->newError(13); //connection must start with the name of a device or end of device
351         list must be terminated with END (not semicolon)
352     }
353     if (!connectionError)
354     {
355         smz->getsymbol(cursym, curname, curint, numstring);
356     }
357     if (!connectionError) erz->newError(24); //must end line in semicolon
358     while (cursym != semicol && cursym != endsym && cursym != eofsym)
359     {
360         smz->getsymbol(cursym, curname, curint, numstring);
361     }
362     if (cursym == semicol)
363     {
364         connectionList();
365     }
366     if (cursym == endsym)
367     {
368         return;
369     }
370 }
371
372 bool parser::newConnection()
373 {
374     //EBNF: con = devicename '.' input '=' devicename['.' output]
375     bool symbolSkip = false;
376     devlink devtype = netz->finddevice(curname);
377     if (devtype != NULL)
378     {
379         connectionInName = curname;
380         smz->getsymbol(cursym, curname, curint, numstring);
381         if (cursym == dot)
382         {
383             smz->getsymbol(cursym, curname, curint, numstring);
384             devtype = netz->finddevice(connectionInName);
385             inplink ilist = netz->findinput(devtype, curname);
386             if (cursym == iosym && ilist != NULL)
387             {
388                 name inputPin = curname;
389                 smz->getsymbol(cursym, curname, curint, numstring);
390                 if (cursym == equals) //SEARCH - you have got to here
391                 {
392                     smz->getsymbol(cursym, curname, curint, numstring);
393                     devtype = netz->finddevice(curname);
394                     if (devtype != NULL)
395                     {
396                         connectionOutName = curname;
397                         switch (devtype ? devtype->kind : baddevice)
398                         {
399                             case 7:
400                                 smz->getsymbol(cursym, curname, curint, numstring);
401                                 if (cursym == dot)

```

```

402     {
403         smz->getsymbol(cursym, curname, curint, numstring);
404         outlink olist = netz->findoutput(devtype, curname);
405         if (cursym == iosym && olist != NULL)
406         {
407             if (ilist->connect==NULL)
408             {
409                 netz->makeconnection(connectionInName, inputPin, connectionOutName,
currname, correctOperation);
410                 return symbolSkip;
411             }
412             else if (ilist->connect==netz->findoutput(devtype, curname))
413             {
414                 namestring repeatedInput = smz->nmz->getnamestring(connectionInName);
415                 namestring repeatedOutputDevice = smz->nmz->getnamestring(
connectionOutName);
416                 namestring repeatedOutputPin = smz->nmz->getnamestring(currname);
417                 namestring repeatedOutput = repeatedOutputDevice + "." +
repeatedOutputPin;
418                 erz->connectionWarning(repeatedInput, repeatedOutput); //generate
warning for repeated connection
419             }
420             else
421             {
422                 erz->newError(37); //attempting to input 2 ouputs into same input
symbolSkip=true;
423             }
424         }
425     }
426     else
427     {
428         erz->newError(34); //Not valid output for dtype
429     }
430 }
431 else
432 {
433     erz->newError(14); //Expect a dot after dtype
434     symbolSkip=true;
435 }
436 break;
437 default:
438     //check the connection is unique
439     if (ilist->connect==NULL)
440     {
441         netz->makeconnection(connectionInName, inputPin, connectionOutName,
blankname, correctOperation);
442         return symbolSkip;
443     }
444     else if (ilist->connect==netz->findoutput(devtype, blankname))
445     {
446         namestring repeatedInput = smz->nmz->getnamestring(connectionInName);
447         namestring repeatedOutput = smz->nmz->getnamestring(connectionOutName);
448         erz->connectionWarning(repeatedInput, repeatedOutput); //generate warning
for repeated connection
449     }
450     else
451     {
452         erz->newError(37); //attempting to input 2 ouputs into same input
symbolSkip=true;
453     }
454 }
455 break;
456 }
457 }
458 else
459 {
460     erz->newError(15); //Device does not exist
symbolSkip=true;
461 }
462 }
463 }
464 else
465 {
466     erz->newError(16); //Must specify output to connect to input with equals sign
symbolSkip=true;
467 }
468 }
469 }
470 else
471 {

```



```

472     erz->newError(17); //specify valid input gate after dot
473     symbolSkip=true;
474 }
475 }
476 else
477 {
478     erz->newError(18); //need to searate connection input with a '.' (or need to specify
input)
479     symbolSkip=true;
480 }
481 }
482 else
483 {
484     erz->newError(19); //Device does not exist
485     symbolSkip=true;
486 }
487 return symbolSkip;
488 }
489
490 void parser::monitorList()
491 {
492     //EBNF: monitors = 'MONITORS' {mon ';' } 'END'
493     bool monitorError;
494     if (!monitorPresent)
495     {
496         smz->getsymbol(cursym, curname, curint, numstring);
497         if (cursym == endsym)
498         {
499             if (!monitorPresent)
500             {
501                 erz->newWarning(1); //No Monitors
502             }
503             return;
504         }
505         else if (cursym == namesym)
506         {
507             monitorError = newMonitor();
508             monitorPresent = true;
509         }
510         else
511         {
512             erz->newError(20); //monitor must start with the name of a device
513         }
514         if (!monitorError)
515         {
516             smz->getsymbol(cursym, curname, curint, numstring);
517         }
518     }
519     while (cursym == semicol)
520     {
521         smz->getsymbol(cursym, curname, curint, numstring);
522         if (cursym == namesym)
523         {
524             monitorError = newMonitor();
525         }
526         else if (cursym == endsym)
527         {
528             return;
529         }
530         else if (cursym == consym | cursym == devsym | cursym == monsym)
531         {
532             erz->newError(32); //Block must be terminated with 'END'
533             return;
534         }
535         else
536         {
537             erz->newError(21); //monitor must start with the name of a device or end of device list
must be terminated with END (not semicolon)
538         }
539         if (!monitorError)
540         {
541             smz->getsymbol(cursym, curname, curint, numstring);
542         }
543     }
544     if (!monitorError) erz->newError(24); //must end line in semicolon
545     while (cursym != semicol && cursym != endsym && cursym != eofsym)

```

```

546 {
547     smz->getsymbol(cursym, curname, curint, numstring);
548 }
549 if (cursym == semicol)
550 {
551     monitorList();
552 }
553 if (cursym == endsym)
554 {
555     return;
556 }
557 }
558
559 bool parser::newMonitor()
560 {
561     //EBNF: mon = devicename['.'output]
562     bool symbolSkip = false;
563     devlink devtype = netz->finddevice(curname);
564     if (devtype != NULL)
565     {
566         monitorName = curname;
567         switch (devtype ? devtype->kind : baddevice)
568         {
569             case 7:
570                 smz->getsymbol(cursym, curname, curint, numstring);
571                 if (cursym == dot)
572                 {
573                     smz->getsymbol(cursym, curname, curint, numstring);
574                     outlink olist = netz->findoutput(devtype, curname);
575                     bool alreadyMonitored = mmz->IsMonitored(olist);
576                     if (!alreadyMonitored)
577                     {
578                         if (cursym == iosym && olist != NULL)
579                         {
580                             mmz->makemonitor(monitorName, curname, correctOperation);
581                             return symbolSkip;
582                         }
583                         else
584                         {
585                             erz->newError(34); //Not valid output for dtype
586                         }
587                     }
588                     else
589                     {
590                         namestring repeatedMonitorDevice = smz->nmz->getnamestring(monitorName);
591                         namestring repeatedMonitorPin = smz->nmz->getnamestring(curname);
592                         namestring repeatedMonitor = repeatedMonitorDevice + "." + repeatedMonitorPin;
593                         erz->monitorWarning(repeatedMonitor); //repeated monitors
594                         if (cursym == iosym && olist != NULL)
595                         {
596                             mmz->makemonitor(monitorName, curname, correctOperation);
597                             return symbolSkip;
598                         }
599                         else
600                         {
601                             erz->newError(34); //Not valid output for dtype
602                         }
603                     }
604                 }
605                 else
606                 {
607                     erz->newError(22); //Expect a dot after dtype
608                     symbolSkip=true;
609                 }
610             default:
611                 outlink olist = netz->findoutput(devtype, blankname);
612                 bool alreadyMonitored = mmz->IsMonitored(olist);
613                 if (!alreadyMonitored)
614                 {
615                     mmz->makemonitor(monitorName, blankname, correctOperation);
616                 }
617                 else
618                 {
619                     namestring repeatedMonitor = smz->nmz->getnamestring(curname);
620                     erz->monitorWarning(repeatedMonitor); //repeated monitors
621                     mmz->makemonitor(monitorName, curname, correctOperation);

```

```

622     }
623     return symbolSkip;
624 }
625 }
626 else
627 {
628     erz->newError(23); //bad device monitor
629     symbolSkip=true;
630 }
631 return symbolSkip;
632 }
633
634 bool parser::isBinary(string numstring)
635 {
636     for (int i=0; i<numstring.length(); i++)
637     {
638         if (numstring[i]!='0' && numstring[i]!='1')
639         {
640             return false;
641         }
642     }
643     return true;
644 }
645
646 parser::parser(network* network_mod, devices* devices_mod, monitor* monitor_mod, scanner*
        scanner_mod, error* error_mod)
647 {
648     netz = network_mod; /* make internal copies of these class pointers */
649     dmz = devices_mod; /* so we can call functions from these classes */
650     mmz = monitor_mod; /* eg. to call makeconnection from the network */
651     smz = scanner_mod; /* class you say: */
652     erz = error_mod; /* netz->makeconnection(i1, i2, o1, o2, ok); */
653     /* any other initialisation you want to do? */
654 }

```

Listing 5: parser.cc

parser.cc was written with joint effort between myself and Tim. I contributed approximately 25% of the code.

A.4 Test Scripts

A.4.1 test.sh

```

1 #!/bin/bash
2 echo "" > error.txt
3 for f in *.gf2
4 do
5     echo "processing $f..."
6     echo "processing $f..." >> test.txt
7     echo -e "r 50\r\nq\r\n" | ../src/logsim $f >> test.txt
8     echo -e '\n' >> test.txt
9 done

```

Listing 6: test.sh

A.4.2 error.sh

```

1 #!/bin/bash
2 echo "" > error.txt
3 for f in *.gf2
4 do
5     echo "processing $f..."
6     echo "processing $f..." >> error.txt
7     echo ../../src/logsim $f >> error.txt
8     echo -e '\n' >> error.txt
9 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

```
1 DEVICES
2 SWITCH S1:0;
3 SWITCH S2:1;
4 NAND G1:2;
5 NAND G2:2;
6 NAND G3:2;
7 NAND G4:2;
8 END
9
10 CONNECTIONS
11 G1.I1 = S1;
12 G1.I2 = S2;
13 G2.I1 = S1;
14 G2.I2 = G1;
15 G3.I1 = G1;
16 G3.I2 = S2;
17 G4.I1 = G2;
18 G4.I2 = G3;
19 END
20
21 MONITORS
22 S1;
23 S2;
24 G4;
25 END
```

Listing 8: xor.gf2

B.1.2 Circuit Diagram

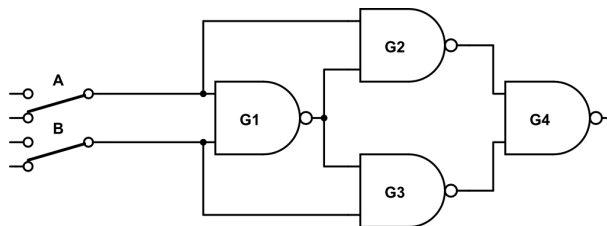


Figure 1: Circuit diagram of an XOR gate implemented using NAND gates

B.2 4-bit Adder

B.2.1 Definition File

```
1 DEVICES
2 /* 4 bit inputs */
3 SWITCH A0:1;
4 SWITCH A1:0;
5 SWITCH A2:0;
6 SWITCH A3:0;
7 SWITCH B0:1;
8 SWITCH B1:0;
9 SWITCH B2:0;
```

```

10 SWITCH B3:0;
11 SWITCH C0:1; /* Carry in */
12 AND AND1:2;
13 AND AND2:2;
14 AND AND3:2;
15 AND AND4:2;
16 AND AND5:2;
17 AND AND6:2;
18 AND AND7:2;
19 AND AND8:2;
20 XOR XOR1;
21 XOR XOR2;
22 XOR XOR3;
23 XOR XOR4;
24 XOR XOR5;
25 XOR XOR6;
26 XOR XOR7;
27 XOR XOR8;
28 OR OR1:2;
29 OR OR2:2;
30 OR OR3:2;
31 OR OR4:2;
32 END
33
34 CONNECTIONS
35 /* LSB adder */
36 XOR1.I1 = A0;
37 XOR1.I2 = B0;
38 AND1.I1 = XOR1;
39 AND1.I2 = C0;
40 AND2.I1 = A0;
41 AND2.I2 = B0;
42 XOR2.I1 = XOR1;
43 XOR2.I2 = C0;
44 OR1.I1 = AND1;
45 OR1.I2 = AND2;
46
47 XOR3.I1 = A1;
48 XOR3.I2 = B1;
49 AND3.I1 = XOR3;
50 AND3.I2 = OR1;
51 AND4.I1 = A1;
52 AND4.I2 = B1;
53 XOR4.I1 = XOR3;
54 XOR4.I2 = OR1;
55 OR2.I1 = AND3;
56 OR2.I2 = AND4;
57
58 XOR5.I1 = A2;
59 XOR5.I2 = B2;
60 AND5.I1 = XOR5;
61 AND5.I2 = OR2;
62 AND6.I1 = A2;
63 AND6.I2 = B2;
64 XOR6.I1 = XOR5;
65 XOR6.I2 = OR2;
66 OR3.I1 = AND5;
67 OR3.I2 = AND6;
68
69 /* MSB Adder */
70 XOR7.I1 = A3;
71 XOR7.I2 = B3;
72 AND7.I1 = XOR7;
73 AND7.I2 = OR3;
74 AND8.I1 = A3;
75 AND8.I2 = B3;
76 XOR8.I1 = XOR7;
77 XOR8.I2 = OR3;
78 OR4.I1 = AND7;
79 OR4.I2 = AND8;
80 END
81
82 MONITORS
83 /* Outputs */
84 XOR2;
85 XOR4;

```

```

86 XOR6;
87 XOR8;
88 OR4; /* Carry out */
89 END

```

Listing 9: 4bitadder.gf2

B.2.2 Circuit Diagram

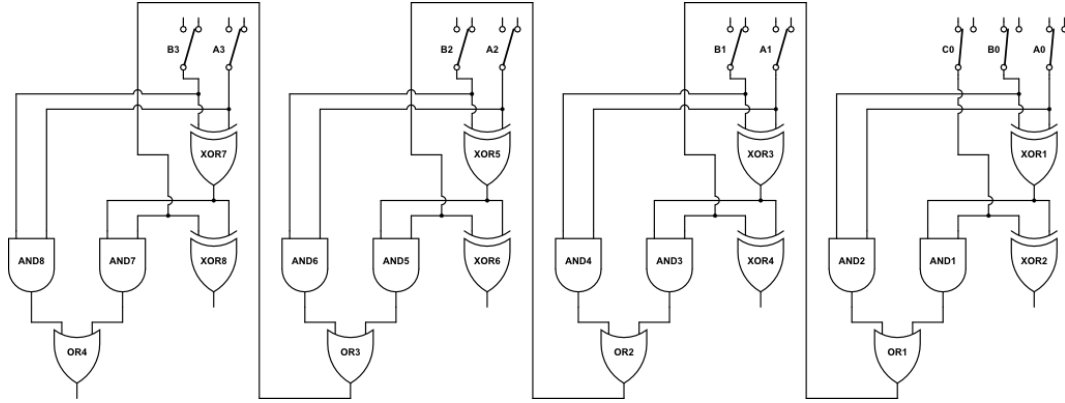


Figure 2: Circuit diagram of a 4-bit adder

B.3 Serial In Parallel Out Shift Register

B.3.1 Definition File

```

1 DEVICES
2 CLOCK CLK1:2;
3 CLOCK CLK2:1;
4 SWITCH S:0; /* Set switch */
5 SWITCH R:0; /* Reset switch */
6 DTYPE D1;
7 DTYPE D2;
8 DTYPE D3;
9 DTYPE D4;
10 END
11
12 CONNECTIONS
13 D1.DATA = CLK1;
14 D2.DATA = D1.Q;
15 D3.DATA = D2.Q;
16 D4.DATA = D3.Q;
17 D1.CLK = CLK2;
18 D2.CLK = CLK2;
19 D3.CLK = CLK2;
20 D4.CLK = CLK2;
21 D1.SET = S;
22 D2.SET = S;
23 D3.SET = S;
24 D4.SET = S;
25 D1.CLEAR = R;
26 D2.CLEAR = R;
27 D3.CLEAR = R;
28 D4.CLEAR = R;
29 END
30
31 MONITORS
32 CLK2;
33 D1.Q;
34 D2.Q;
35 D3.Q;
36 D4.Q;
37 END

```

B.3.2 Circuit Diagram

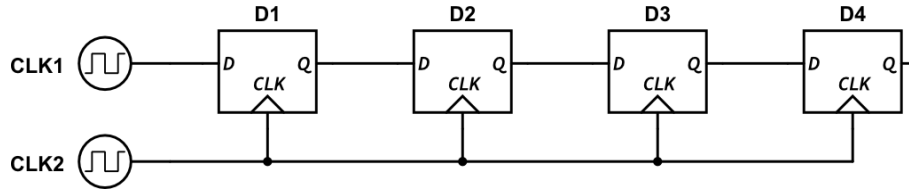


Figure 3: Circuit diagram of a serial in parallel out shift register

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. 3 was the closest achievable.

B.4 Gated D Latch

B.4.1 Definition File

```

1 DEVICES
2 CLOCK CLK1:1;
3 CLOCK CLK2:2;
4 NAND G1:1;
5 AND G2:2;
6 AND G3:2;
7 NOR G4:2;
8 NOR G5:2;
9 END
10
11 CONNECTIONS
12 G1.I1 = CLK1;
13 G2.I1 = G1;
14 G2.I2 = CLK2;
15 G3.I1 = CLK2;
16 G3.I2 = CLK1;
17 G4.I1 = G2;
18 G4.I2 = G5;
19 G5.I1 = G4;
20 G5.I2 = G3;
21 END
22
23 MONITORS
24 CLK1; /* D */
25 CLK2; /* E */
26 G4; /* Q */
27 G5; /* QBAR */
28 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. 4.



C EBNF

```
1 specfile = devices connections monitors
2
3 devices = 'DEVICES' dev ';' {dev ';' } 'END'
4 connections = 'CONNECTIONS' {con ';' } 'END'
5 monitors = 'MONITORS' {mon ';' } 'END'
6
7 dev = clock|switch|gate|dtype|xor|siggen
8 con = devicename '.' input '=' devicename[ '.' output]
9 mon = devicename[ '.' output]
10
11 devicename = letter { '_' | letter | digit}
12 input = 'I' ( '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9' |
13           '10' | '11' | '12' | '13' | '14' | '15' | '16' ) | 'DATA' | 'CLK' | 'SET' | 'CLEAR'
14 output = 'Q' [ 'BAR' ]
15
16 clock = 'CLOCK' devicename ':' digit { digit}
17 switch = 'SWITCH' devicename ':' ( '0' | '1' )
18 gate = ( 'AND' | 'NAND' | 'OR' | 'NOR' ) devicename ':' ( '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' |
19           '9' | '10' | '11' | '12' | '13' | '14' | '15' | '16' )
20 dtype = 'DTYPE' devicename
21 xor = 'XOR' devicename
22 siggen = 'SIGGEN' devicename ':' ( '0' | '1' ) { '0' | '1' }
```

Listing 13: EBNF

D User Guide

Opening files: To open a definition file, click the **File** menu followed by the **Open** option. You will be presented with a file selection dialogue. The file selection dialogue will only show definition files (Files with the **.gf2** file extension). Upon selecting a file, any errors in the definition file will be written to the message window, otherwise the Logic Simulator is ready to use.

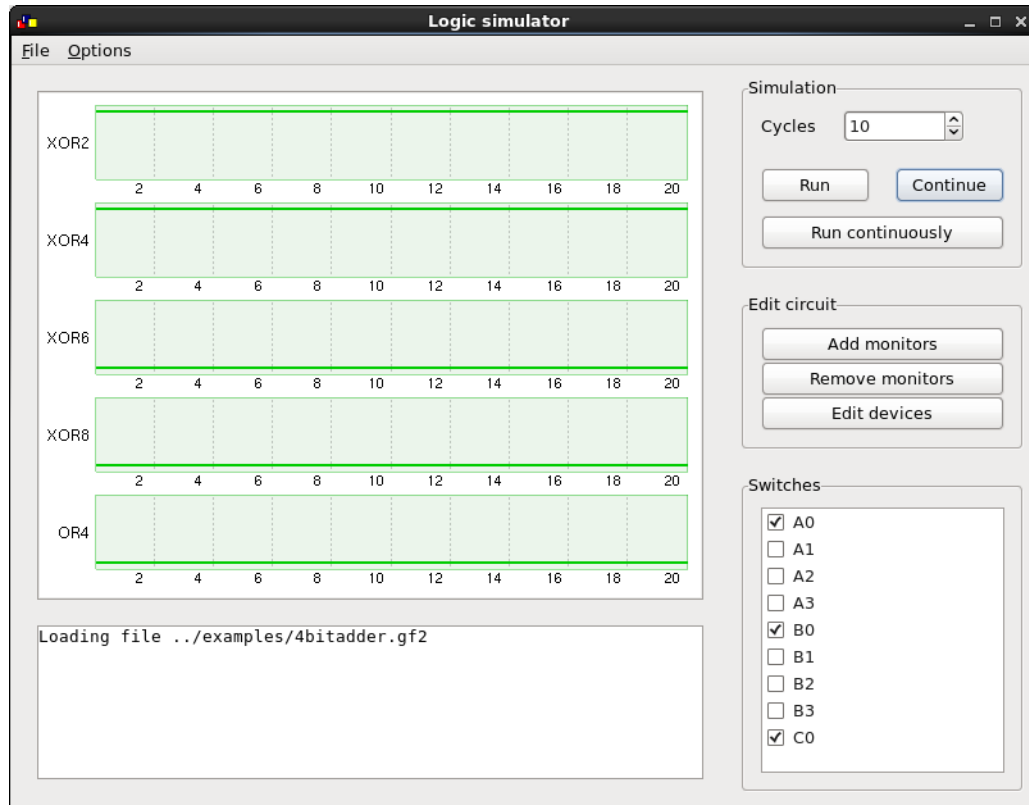


Figure 6: The view upon running a simulation

Running simulations: In order to run a simulation you must first enter a number of cycles you wish the simulation to run for (default is 10) then press the run button. The monitored signals will be displayed in the left display panel. You may choose to continue the simulation by pressing the **Continue** button, or run the simulation continuously by pressing the **Continuous Simulation** button.

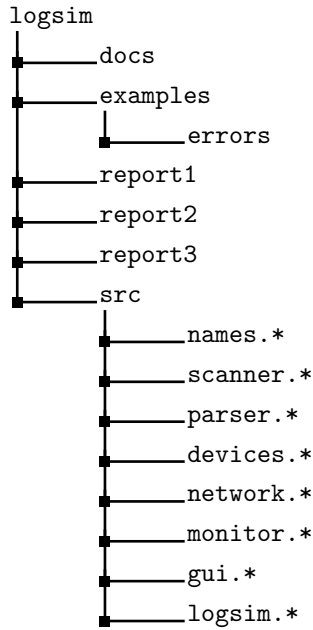
Adding or removing monitors: You can edit the signals that are monitored and displayed in the main window. To add monitors, click the **Add monitors** button and select the monitor, or monitors, you wish to add followed by the **OK** button. To remove monitors, press the **Remove monitors** button and select the monitor, or monitors, you wish to remove followed by the **OK** button.

Editing devices: To edit devices, click the **Edit devices** button. From the Edit devices dialogue you can change the device's name, type and number of inputs (if applicable). You can also change the inputs to, or outputs from a device.

Changing switch states: If your circuit contains any switches, you can change the state of the switch by changing the state of the check box beside its name.

Changing simulator settings: You can change certain settings relating to the operation of the logic simulator. Click the **Options** menu and from here you can change or reset settings as well as adjust the continuous run speed. By clicking **Edit options** you can change the trace options and debugging options.

E File Listing



docs contains documentation relevant to our logic simulator such as: EBNF, reserved words, and our Gantt chart.

examples contains our example definition files as listed in Appendix B as well as the test shell script listed in Appendix A.4.

errors contains definition files which contain deliberate errors and are used to test our error checking functionality. In addition this folder contains the error.sh shell script listed in Appendix A.4.

report1, **report2** and **report3** contain our first, second and final report respectively.

src contains the source code for our logic simulator. The functionality of the major classes is outlined beneath.

names stores a list of all the words used within a definition file, and methods to manipulate them. It is initialised with only the reserved words, but can be populated as a definition file is read.

scanner reads through the definition file, character by character, and is able to return complete symbols to the parser. It is able to return the internal representation of a symbol, the type of symbol and optionally the value.

parser analyses the definition file as it is read in according to the rules laid out in our EBNF. It is then able to create devices, connections and monitors that are laid out in the definition file.

devices is used to create and execute devices.

network stores information about the devices, including the connections between each device.

monitor implements signal monitors, which are used to display the points of interest, as determined from the definition file, in the GUI.

gui provides a graphical user interface for the user, and allows for advanced features such as circuit and monitor editing without having to edit the definition file. We have implemented the GUI with a modular approach, and as such there are several gui related classes e.g. gui-canvas, gui-id, gui-misc, etc.

logsim ties all the other classes together and allows the user to fully simulate logic circuits.

F Gantt Chart

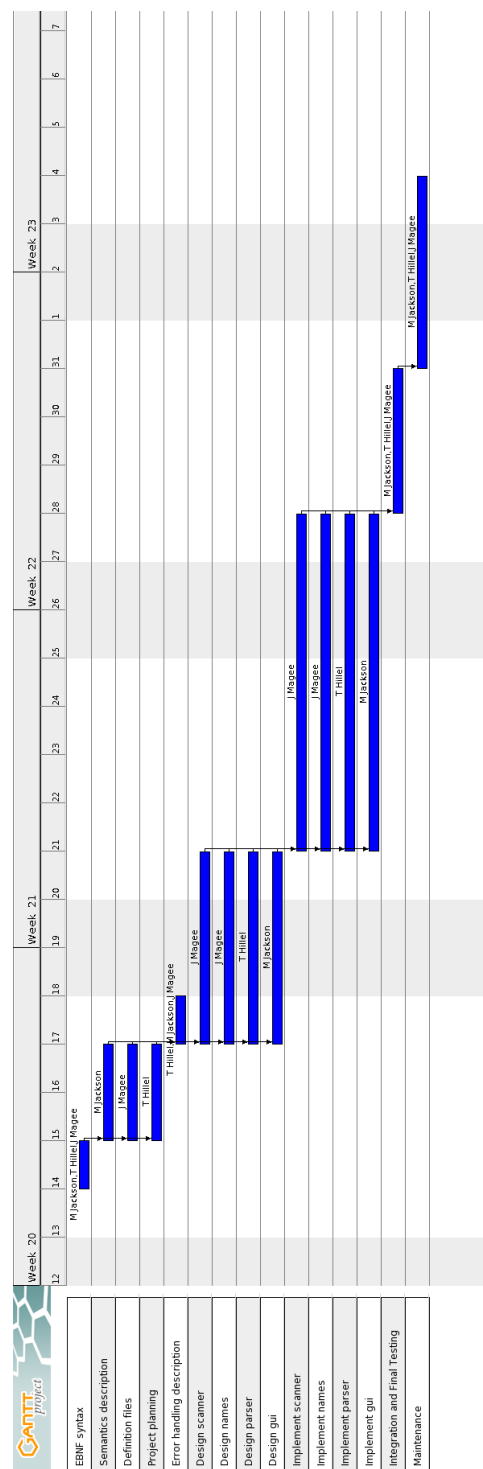


Figure 7: Gantt chart showing our development cycle