

IIA GF2 Software: 2nd Interim Report

Jamie Magee (jam96) - Team 8

1 Code Listings

1.1 Names Class

1.1.1 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 = 33;
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

1.1.2 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("I1"); //12
26     namelist.push_back("I2"); //13
27     namelist.push_back("I3"); //14
28     namelist.push_back("I4"); //15
29     namelist.push_back("I5"); //16
30     namelist.push_back("I6"); //17
31     namelist.push_back("I7"); //18
32     namelist.push_back("I8"); //19
33     namelist.push_back("I9"); //20
34     namelist.push_back("I10"); //21
35     namelist.push_back("I11"); //22
36     namelist.push_back("I12"); //23
37     namelist.push_back("I13"); //24
38     namelist.push_back("I14"); //25
39     namelist.push_back("I15"); //26
40     namelist.push_back("I16"); //27
41     namelist.push_back("DATA"); //28
42     namelist.push_back("CLK"); //29
43     namelist.push_back("SET"); //30
44     namelist.push_back("CLEAR"); //31
45     namelist.push_back("Q"); //32
46     namelist.push_back("QBAR"); //33
47 }
48
49 name names::lookup(namestring str)
50 {
51     if (cvtname(str) == blankname)
52     {
53         namelist.push_back(str); //Insert new string
54         return namelist.size() - 1; //Return new strings internal name
55     }
56     else
57     {
58         return cvtname(str);
59     }
60 }
61
62 name names::cvtname(namestring str)
63 {
64     if (str == "") return blankname;
65     for (name id = 0; id < namelist.size(); id++)
66     {
67         if (namelist[id] == str) return id; //Linear search of namelist vector
68     }
69     return blankname;
70 }
71
72 void names::writename(name id)
73 {
74     if (id == blankname) cout << "blankname";
75     else if (id > blankname && id < namelist.size()) cout << namelist[id];
76     else cout << "Incorrect id";
77 }
78
79 int names::namelength(name id)
80 {
81     if (id > blankname && id < namelist.size()) return namelist[id].length();
82     else return blankname;
83 }

```

```

84
85 namestring names::getnamestring(name id)
86 {
87     if (id > blankname && id < namelist.size()) return namelist[id];
88     else return "";
89 }

```

Listing 2: names.cc

1.2 Scanner Class

1.2.1 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,
13     equals, dot, badsym, eofsym} symbol;
14
15 class scanner
16 {
17     public:
18         symbol s;
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); //Return symbol value (if it's a number)
27         void writelineerror();
28     private:
29         ifstream inf; //Input file
30         names* nmz; //Pointer to instance of names class
31         char curch; //Current input character
32         char prevch; //Previous input character. Used for finding line end
33         bool eofile; //True for end of file
34         bool ok; //True if the file has been opened correctly
35         int linenum; //Number of lines in definition file
36         int cursymlen; //Length of current symbol. Used for error printing
37         string line; //Current line contents. Used for error printing
38
39         void getch(); //Gets next input character
40         void getnumber(int& number); //Reads number from file
41         void getname(name& id); //Reads name from file
42         string getline(); //Reads the line
43         void skipspaces(); //Skips spaces
44         void skipcomments(); //Skips comments
45 };
46
47 #endif

```

Listing 3: scanner.h

1.2.2 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 }
19
20 scanner::~scanner()
21 {
22     inf.close(); //Close file
23 }
24
25 void scanner::getsymbol(symbol& s, name& id, int& num)
26 {
27     s = badsym;
28     cursymlen = 0;
29     skipspaces();
30     if (eofile) s = eofsym;
31     else
32     {
33         if (isdigit(curch))
34         {
35             s = numsym;
36             getnumber(num);
37         }
38         else
39         {
40             if (isalpha(curch))
41             {
42                 getname(id);
43                 if (id == 0) s = devsym;
44                 else if (id == 1) s = consym;
45                 else if (id == 2) s = monsym;
46                 else if (id == 3) s = endsym;
47                 else if (id > 3 && id < 12) s = classsym;
48                 else if (id > 11 && id < 34) s = iosym;
49                 else s = namesym;
50             }
51             else
52             {
53                 switch (curch)
54                 {
55                     case '=':
56                         s = equals;
57                         getch();
58                         break;
59                     case ';':
60                         s = semicol;
61                         getch();
62                         break;
63                     case ':':
64                         s = colon;
65                         getch();
66                         break;
67                     case '.':
68                         s = dot;
69                         getch();
70                         break;
71                     case '/':
72                         getch();
73                         if (curch == '*')
74                         {
75                             getch();
76                             skipcomments();
77                             getsymbol(s, id, num);
78                         }
79                         break;
80                     default:
81                         s = badsym;
82                         getch();
83                         break;
84                 }
85                 cursymlen = 1;

```

```

86     }
87 }
88 }
89 }
90
91 void scanner::writelineerror()
92 {
93     string errorptr;
94     for (int i = 0; i < ((int)line.length() - cursymlen); i++)
95     {
96         errorptr.push_back(' ');
97     }
98     errorptr.push_back('^');
99     cout << "Line " << linenum << ":" << endl;
100    cout << getline() << endl; //Outputs current line
101    cout << errorptr << endl; //Outputs a caret at the error
102 }
103
104 void scanner::getch()
105 {
106     prevch = curch;
107     eofile = (inf.get(curch) == 0); //get next character
108     if (prevch == '\n') //If eoline, clear the currently stored line
109     {
110         linenum++;
111         line.clear();
112     }
113     else if (prevch != '\r') //If we're not at the end of a line, add the char to the line string
114     {
115         line.push_back(prevch);
116     }
117 }
118
119 void scanner::getnumber(int& number)
120 {
121     number = 0;
122     cursymlen = 0;
123     while (isdigit(curch) && !eofile)
124     {
125         number *= 10;
126         number += (int(curch) - int('0'));
127         cursymlen++;
128         getch();
129     }
130 }
131
132 void scanner::getname(name& id)
133 {
134     namestring str;
135     cursymlen = 0;
136     while (isalnum(curch) && !eofile)
137     {
138         str.push_back(curch);
139         cursymlen++;
140         getch();
141     }
142     id = nmz->lookup(str);
143 }
144
145 void scanner::skipspaces()
146 {
147     while (isspace(curch) || curch == '\n')
148     {
149         getch();
150         if (eofile) break;
151     }
152 }
153
154 void scanner::skipcomments()
155 {
156     while (!(prevch == '*' && curch == '/'))
157     {
158         getch();
159         if (eofile)
160         {
161             cout << "Reached end of file before comment was terminated" << endl;
162             break;
163         }
164     }

```

```

165     getch(); //Get to next useful char
166 }
167
168 string scanner::getline()
169 {
170     if (s != semicol)
171     {
172         while (curch != ';' && !eofile && curch != '\n')
173         {
174             getch();
175         }
176         if (curch != '\n' && curch != '\r')
177         {
178             line.push_back(curch);
179         }
180     }
181     return line;
182 }

```

Listing 4: scanner.cc

1.3 Parser Class

1.3.1 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     cursym = badsym;
14     while (cursym != eofsym)
15     {
16         if (cursym != devsym && cursym != consym && cursym != monsym)
17         {
18             smz->getsymbol(cursym, curname, curint);
19         }
20         if (cursym == devsym)
21         {
22             if (deviceDone)
23             {
24                 erz->newError(25); //Must only be one devices list
25             }
26             deviceDone = true;
27             deviceList();
28         }
29         else if (cursym == consym)
30         {
31             if (!deviceDone)
32             {
33                 erz->newError(0); //must have device list first
34             }
35             if (connectionDone)
36             {
37                 erz->newError(28); //Must only be one connections list
38             }
39             connectionDone = true;
40             connectionList();
41         }
42         else if (cursym == monsym)
43         {
44             if (!deviceDone | !connectionDone)
45             {
46                 erz->newError(2); //Must have monitor list last
47             }
48             if (monitorDone)
49             {
50

```

```

51     erz->newError(29); //Must only be one Monitors list
52 }
53 monitorDone = true;
54 monitorList();
55 }
56 else if (cursym != eofsym)
57 {
58     while (cursym != devsym && cursym != consym && cursym != monsym && cursym != eofsym)
59     {
60         smz->getsymbol(cursym, curname, curint);
61         erz->countSymbols();
62     }
63     erz->symbolError(deviceDone, connectionDone, monitorDone);
64 }
65 }
66 if (!deviceDone)
67 {
68     erz->newError(26); //There must be a DEVICES block, it may not have been initialised properly
69 }
70 if (!connectionDone)
71 {
72     erz->newError(30); //There must be a CONNECTIONS block, it may not have been initialised properly
73 }
74 if (!monitorDone)
75 {
76     erz->newError(31); //There must be a MONITORS block, it may not have been initialised properly
77 }
78 netz->checknetwork(correctOperation);
79 anyErrors = erz->anyErrors();
80 return (correctOperation && !anyErrors);
81 }
82
83 void parser::deviceList()
84 {
85     //EBNF: devices = 'DEVICES' dev {';' dev} ';' 'END'
86     bool deviceError;
87     if (!devicePresent)
88     {
89         smz->getsymbol(cursym, curname, curint);
90         if (cursym == classsym)
91         {
92             deviceError = newDevice(curname);
93             devicePresent = true;
94         }
95         else if (cursym == endsym)
96         {
97             erz->newError(3); //must have at least one device
98             return;
99         }
100     else
101     {
102         erz->newError(4); //need a device type
103     }
104     if (!deviceError)
105     {
106         smz->getsymbol(cursym, curname, curint);
107     }
108 }
109 while (cursym == semicol)
110 {
111     smz->getsymbol(cursym, curname, curint);
112     if (cursym == classsym)
113     {
114         deviceError = newDevice(curname);
115     }
116     else if (cursym == endsym)
117     {
118         return;
119     }
120     else if (cursym == consym | cursym == devsym | cursym == monsym)
121     {
122         erz->newError(32); //Block must be terminated with 'END'
123         return;
124     }
125     else
126     {
127         erz->newError(5); //Expecting device name or END after semicolon (device name must start with
128         letter)
129     }

```

```

129     if (!deviceError)
130     {
131         smz->getsymbol(cursym, curname, curint);
132     }
133 }
134 if (!deviceError) erz->newError(24); //must end line in semicolon
135 while (cursym != semicol && cursym != endsym && cursym != eofsym)
136 {
137     smz->getsymbol(cursym, curname, curint);
138 }
139 if (cursym == semicol)
140 {
141     deviceList();
142 }
143 if (cursym == endsym)
144 {
145     return;
146 }
147 }
148
149
150 bool parser::newDevice(int deviceType)
151 {
152     //EBNF: dev = clock|switch|gate|dtype|xor
153     bool errorOccurance = false;
154     smz->getsymbol(cursym, curname, curint);
155     if (cursym == namesym)
156     {
157         devlink nameCheck = netz->finddevice(curname);
158         if (nameCheck==NULL)
159         {
160             name devName = curname;
161             if (deviceType == 10)
162             {
163                 dmz->makedevice(dtype, devName, 0, correctOperation); //create DTYPE with name devName
164                 return errorOccurance;
165             }
166             if (deviceType == 11)
167             {
168                 dmz->makedevice(xorgate, devName, 2, correctOperation); //create XOR with name devName
169                 return errorOccurance;
170             }
171             smz->getsymbol(cursym, curname, curint);
172             if (cursym == colon)
173             {
174                 smz->getsymbol(cursym, curname, curint);
175                 if (cursym == numsym)
176                 {
177                     switch (deviceType)
178                     {
179                         case 4:
180                             if (curint > 0)
181                             {
182                                 dmz->makedevice(aclock, devName, curint, correctOperation); //create clock with
curint and devName
183                             }
184                             else
185                             {
186                                 erz->newError(6); //clock must have number greater than 0
187                                 errorOccurance=true;
188                             }
189                             break;
190                         case 5:
191                             if (curint == 1 || curint == 0)
192                             {
193                                 dmz->makedevice(asmith, devName, curint, correctOperation); //create switch with
curint and devName
194                             }
195                             else
196                             {
197                                 erz->newError(7); //switch must have either 0 or 1
198                                 errorOccurance=true;
199                             }
200                             break;
201                         case 6:
202                         case 7:
203                         case 8:
204                         case 9:
205                             if (curint > 0 && curint < 17)

```



```

206         {
207             switch (deviceType)
208             {
209                 case 6:
210                     dmz->makedevice(andgate, devName, curint, correctOperation); //create and gate
211                     with curint and devName
212                     break;
213                 case 7:
214                     dmz->makedevice(nandgate, devName, curint, correctOperation); //create nand gate
215                     with curint and devName
216                     break;
217                 case 8:
218                     dmz->makedevice(orgate, devName, curint, correctOperation); //create or gate with
219                     curint and devName
220                     break;
221                 case 9:
222                     dmz->makedevice(norgate, devName, curint, correctOperation); //create nor gate
223                     with curint and devName
224                     break;
225                 default:
226                     cout << "How on earth have you managed to get here?" << endl;
227             }
228         }
229         else
230         {
231             erz->newError(8); //must have between 1 and 16 inputs to a GATE
232             errorOccurance=true;
233         }
234         break;
235     default:
236         cout << "Please do not deduct marks if this message is displayed" << endl;
237     }
238     return errorOccurance;
239 }
240 else
241 {
242     erz->newError(9); //clock needs clock cycle number
243     errorOccurance=true;
244 }
245 }
246 else
247 {
248     erz->newError(10); //need colon after name for CLOCK/SWITCH/GATE type
249     errorOccurance=true;
250 }
251 }
252 else
253 {
254     erz->newError(34); //attempting to give two devices the same name, choose an alternative name
255     errorOccurance=true;
256 }
257 }
258 else if (cursym!=badsym)
259 {
260     erz->newError(33); //using reserved word as device name
261     errorOccurance=true;
262 }
263 else
264 {
265     erz->newError(11); //name must begin with letter and only containing letter number and _
266     errorOccurance=true;
267 }
268 return errorOccurance;
269 }
270
271 void parser::connectionList()
272 {
273     //EBNF: connections = 'CONNECTIONS' {con ';' } 'END'
274     bool connectionError;
275     if (!connectionPresent)
276     {
277         smz->getsymbol(cursym, curname, curint);
278         if (cursym == endsym)
279         {
280             if (!connectionPresent)
281             {
282                 erz->newWarning(0); //No Connections
283             }
284             return;
285         }
286     }

```

```

281     }
282     else if (cursym == namesym)
283     {
284         connectionError = newConnection();
285         connectionPresent = true;
286     }
287     else
288     {
289         erz->newError(12); //connection must start with the name of a device
290     }
291     if (!connectionError)
292     {
293         smz->getsymbol(cursym, curname, curint);
294     }
295 }
296 while (cursym == semicol)
297 {
298     smz->getsymbol(cursym, curname, curint);
299     if (cursym == namesym)
300     {
301         connectionError = newConnection();
302     }
303     else if (cursym == endsym)
304     {
305         return;
306     }
307     else if (cursym == consym | cursym == devsym | cursym == monsym)
308     {
309         erz->newError(32); //Block must be terminated with 'END'
310         return;
311     }
312     else
313     {
314         erz->newError(13); //connection must start with the name of a device or end of device list must
315         be terminated with END (not semicolon)
316     }
317     if (!connectionError)
318     {
319         smz->getsymbol(cursym, curname, curint);
320     }
321 }
322 if (!connectionError) erz->newError(24); //must end line in semicolon
323 while (cursym != semicol && cursym != endsym && cursym != eofsym)
324 {
325     smz->getsymbol(cursym, curname, curint);
326 }
327 if (cursym == semicol)
328 {
329     connectionList();
330 }
331 if (cursym == endsym)
332 {
333     return;
334 }
335 }
336
337 bool parser::newConnection()
338 {
339     //EBNF: con = devicename '.' input '=' devicename ['.' output]
340     bool errorOccurance = false;
341     devlink devtype = netz->finddevice(curname);
342     if (devtype != NULL)
343     {
344         connectionInName = curname;
345         smz->getsymbol(cursym, curname, curint);
346         if (cursym == dot)
347         {
348             smz->getsymbol(cursym, curname, curint);
349             devtype = netz->finddevice(connectionInName);
350             inplink ilist = netz->findinput(devtype, curname);
351             if (cursym == iosym && ilist != NULL)
352             {
353                 name inputPin = curname;
354                 smz->getsymbol(cursym, curname, curint);
355                 if (cursym == equals) //SEARCH - you have got to here
356                 {
357                     smz->getsymbol(cursym, curname, curint);
358                     devtype = netz->finddevice(curname);
359                     if (devtype != NULL)

```

```

359     {
360         connectionOutName = curname;
361         switch (devtype ? devtype->kind : baddevice)
362         {
363             case 7:
364                 smz->getsymbol(cursym, curname, curint);
365                 if (cursym == dot)
366                 {
367                     smz->getsymbol(cursym, curname, curint);
368                     outplink olist = netz->findoutput(devtype, curname);
369                     if (cursym == iosym && olist != NULL)
370                     {
371                         netz->makeconnection(connectionInName, inputPin, connectionOutName, curname,
correctOperation);
372                         return errorOccurance;
373                     }
374                     else
375                     {
376                         erz->newError(34); //Not valid output for dtype
377                     }
378                 }
379                 else
380                 {
381                     erz->newError(14); //Expect a dot after dtype
382                     errorOccurance=true;
383                 }
384             default:
385                 netz->makeconnection(connectionInName, inputPin, connectionOutName, blankname,
correctOperation);
386                 return errorOccurance;
387             }
388         }
389         else
390         {
391             erz->newError(15); //Device does not exist
392             errorOccurance=true;
393         }
394     }
395     else
396     {
397         erz->newError(16); //Must specify output to connect to input with equals sign
398         errorOccurance=true;
399     }
400 }
401 else
402 {
403     erz->newError(17); //specify valid input gate after dot
404     errorOccurance=true;
405 }
406 }
407 else
408 {
409     erz->newError(18); //need to seperate connection input with a '.' (or need to specify input)
410     errorOccurance=true;
411 }
412 }
413 else
414 {
415     erz->newError(19); //Device does not exist
416     errorOccurance=true;
417 }
418 return errorOccurance;
419 }
420
421 void parser::monitorList()
422 {
423     //EBNF: monitors = 'MONITORS' {mon ';' } 'END'
424     bool monitorError;
425     if (!monitorPresent)
426     {
427         smz->getsymbol(cursym, curname, curint);
428         if (cursym == endsym)
429         {
430             if (!monitorPresent)
431             {
432                 erz->newWarning(1); //No Monitors
433             }
434             return;
435         }

```

```

436     else if (cursym == namesym)
437     {
438         monitorError = newMonitor();
439         monitorPresent = true;
440     }
441     else
442     {
443         erz->newError(20); //monitor must start with the name of a device
444     }
445     if (!monitorError)
446     {
447         smz->getsymbol(cursym, curname, curint);
448     }
449 }
450 while (cursym == semicol)
451 {
452     smz->getsymbol(cursym, curname, curint);
453     if (cursym == namesym)
454     {
455         monitorError = newMonitor();
456     }
457     else if (cursym == endsym)
458     {
459         return;
460     }
461     else if (cursym == consym | cursym == devsym | cursym == monsym)
462     {
463         erz->newError(32); //Block must be terminated with 'END'
464         return;
465     }
466     else
467     {
468         erz->newError(21); //monitor must start with the name of a device or end of device list must be
            terminated with END (not semicolon)
469     }
470     if (!monitorError)
471     {
472         smz->getsymbol(cursym, curname, curint);
473     }
474 }
475 if (!monitorError) erz->newError(24); //must end line in semicolon
476 while (cursym != semicol && cursym != endsym && cursym != eofsym)
477 {
478     smz->getsymbol(cursym, curname, curint);
479 }
480 if (cursym == semicol)
481 {
482     monitorList();
483 }
484 if (cursym == endsym)
485 {
486     return;
487 }
488 }
489
490 bool parser::newMonitor()
491 {
492     //EBNF: mon = devicename['.'output]
493     bool errorOccurance = false;
494     devlink devtype = netz->finddevice(curname);
495     if (devtype != NULL)
496     {
497         monitorName = curname;
498         switch (devtype ? devtype->kind : baddevice)
499         {
500             case 7:
501                 smz->getsymbol(cursym, curname, curint);
502                 if (cursym == dot)
503                 {
504                     smz->getsymbol(cursym, curname, curint);
505                     outplink olist = netz->findoutput(devtype, curname);
506                     if (cursym == iosym && olist != NULL)
507                     {
508                         mmz->makemonitor(monitorName, curname, correctOperation);
509                         return errorOccurance;
510                     }
511                     else
512                     {
513                         erz->newError(34); //Not valid output for dtype

```

```

514     }
515   }
516   else
517   {
518     erz->newError(22); //Expect a dot after dtype
519     errorOccurance=true;
520   }
521   default :
522     mmz->makemonitor(monitorName, blankname, correctOperation);
523     return errorOccurance;
524   }
525 }
526 else
527 {
528   erz->newError(23);
529   errorOccurance=true;
530 }
531 return errorOccurance;
532 }
533
534 parser::parser(network* network_mod, devices* devices_mod, monitor* monitor_mod, scanner*
535               scanner_mod, error* error_mod)
536 {
537   netz = network_mod; /* make internal copies of these class pointers */
538   dmz = devices_mod;  /* so we can call functions from these classes */
539   mmz = monitor_mod;  /* eg. to call makeconnection from the network */
540   smz = scanner_mod;  /* class you say: */
541   erz = error_mod; /* netz->makeconnection(i1, i2, o1, o2, ok); */
542   /* any other initialisation you want to do? */
543 }

```

Listing 5: parser.cc

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

2 Test Definition Files

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

2.1 XOR Gate

2.1.1 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

```

2.1.2 Circuit Diagram

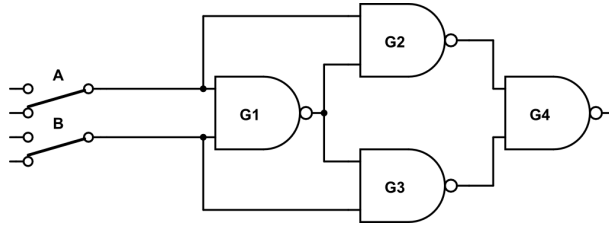


Figure 1: Circuit diagram of an XOR gate implemented using NAND gates

2.2 4-bit Adder

2.2.1 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 7: 4bitadder.gf2

2.2.2 Circuit Diagram

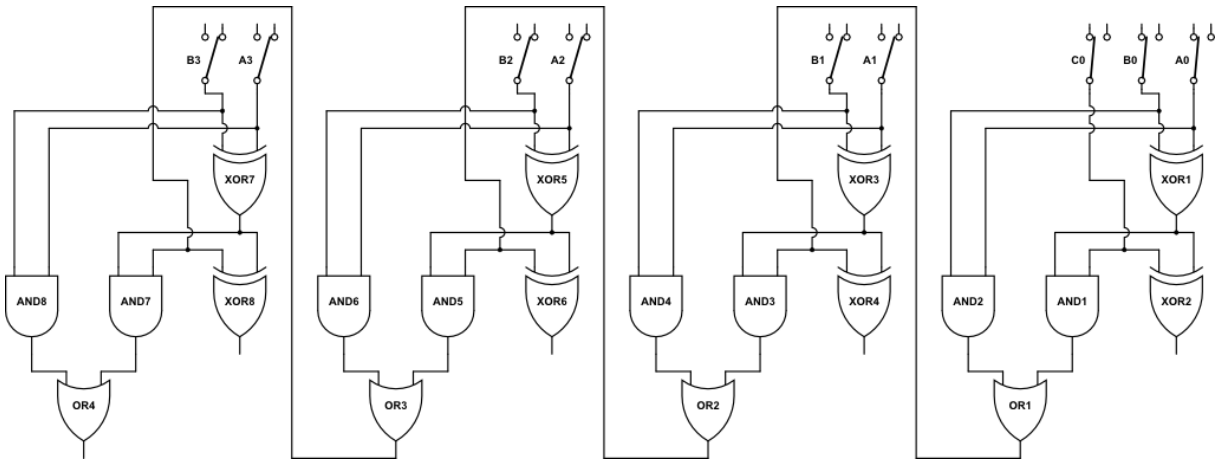


Figure 2: Circuit diagram of a 4-bit adder

2.3 Serial In Parallel Out Shift Register

2.3.1 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

```

Listing 8: sipo.gf2

2.3.2 Circuit Diagram

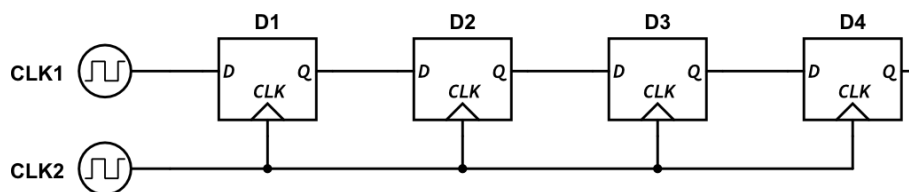


Figure 3: Circuit diagram of a serial in parallel out shift register

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.

2.4 Gated D Latch

2.4.1 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 9: sipo.gf2

2.4.2 Circuit Diagram

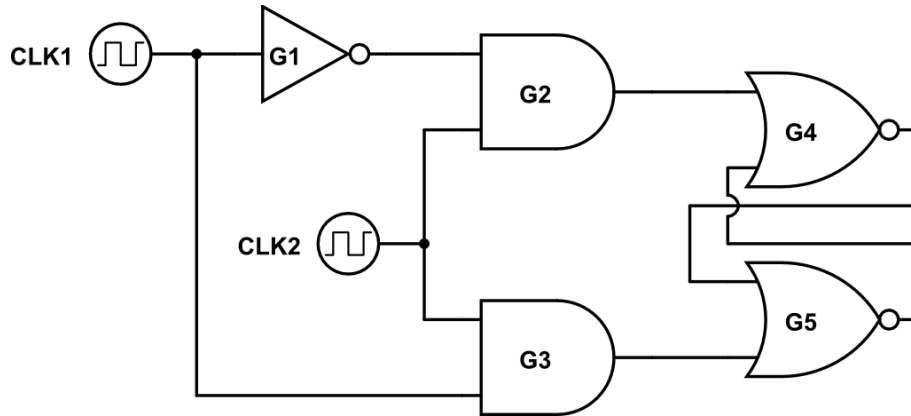


Figure 4: Circuit diagram of a Gated D Latch

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.

3 User Guide

To start the logic simulator, open a terminal window and browse to the `src` folder. Start the application by typing `./logsim` followed by the return key. You will then be presented with the default view.

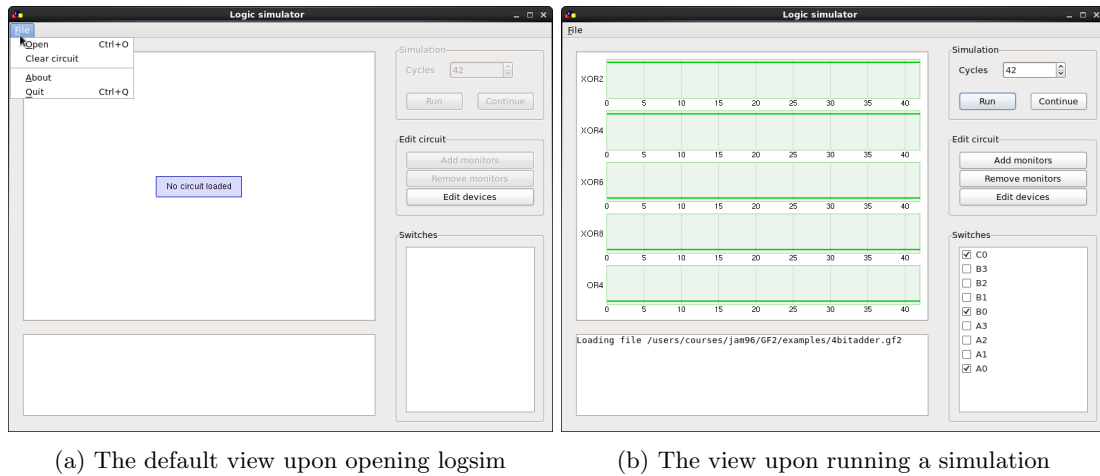


Figure 5: The logsim GUI

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.

In order to run a simulation you must first enter a number of cycles you wish the simulation to run for (default is 42) 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.

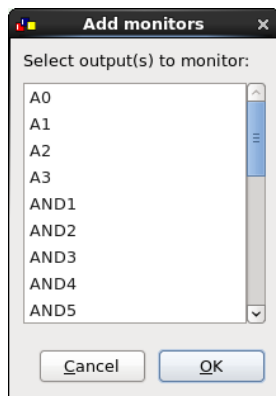


Figure 6: Add monitors dialogue

You can also edit the monitors from within the logic simulator. 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.

In addition, 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.

If you wish to edit your devices you can also do so from within the GUI. 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.

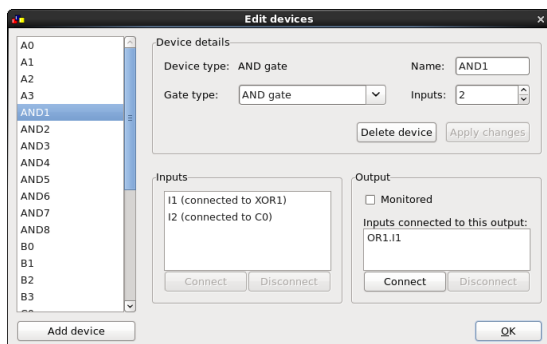


Figure 7: Edit devices dialogue