IIA GF2 Software: 2nd Interim Report

Jamie Magee (jam96) - Team 8

1 Code Listings

1.1 Names Class

1.1.1 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

1.1.2 names.cc

```
#include "names.h"
#include <iostream>
#include <string>
#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,
       equals, dot, badsym, eofsym} symbol;
13
14 class scanner
15 {
16   public:
17     symbol s;
18
19     scanner(names* names_mod,    //Pointer to names class
20         const char* defname,  //Name of file being read
21         bool& ok);         //True of file has been opened correctly
22     ~scanner();            //Destructor
23     void getsymbol(symbol& s,    //Symbol type read
24               name& id,       //Return symbol name (if it has one)
25               int& num);      //Return symbol value (if it's a number)
26     void writelineerror();
27
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)
```

```
{
  nmz = names_mod;
  ok = 1;
  inf.open(defname);  //Open file
  if (!inf)
  {
    cout << "Error: cannot open file for reading" << endl;
    ok = 0;
  }
  eofile = (inf.get(curch) == 0); //Get first character
  linenum = 1;
}

scanner::~scanner()
{
  inf.close();  //Close file
}

void scanner::getsymbol(symbol& s, name& id, int& num)
{
  s = badsym;
  cursymlen = 0;
  skipspaces();
  if (eofile) s = eofsym;
  else
  {
    if (isdigit(curch))
    {
      s = numsym;
      getnumber(num);
    }
    else
    {
      if (isalpha(curch))
      {
        getname(id);
        if (id == 0) s = devsym;
        else if (id == 1) s = consym;
        else if (id == 2) s = monsym;
        else if (id == 3) s = endsym;
        else if (id > 3 && id < 12) s = classsym;
        else if (id > 11 && id < 34) s = iosym;
        else s = namesym;
      }
      else
      {
        switch (curch)
        {
          case '=':
            s = equals;
            getch();
            break;
          case ';':
            s = semicol;
            getch();
            break;
          case ':':
            s = colon;
            getch();
            break;
          case '.':
            s = dot;
            getch();
            break;
          case '/':
            getch();
            if (curch == '*')
            {
              getch();
              skipcomments();
              getsymbol(s, id, num);
            }
            break;
          default:
            s = badsym;
            getch();
            break;
        }
        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       }
40       connectionDone = true;
41       connectionList();
42     }
43     else if (cursym == monsym)
44     {
45       if (!deviceDone | !connectionDone)
46       {
47         erz->newError(2); //Must have monitor list last
48       }
49       if (monitorDone)
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
      letter)
128     }
```

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

```
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
       be terminated with END (not semicolon)
315       }
316       if (!connectionError)
317       {
318         smz->getsymbol(cursym, curname, curint);
319       }
320     }
321     if (!connectionError) erz->newError(24);//must end line in semicolon
322     while (cursym != semicol && cursym != endsym && cursym != eofsym)
323     {
324       smz->getsymbol(cursym, curname, curint);
325     }
326     if (cursym == semicol)
327     {
328       connectionList();
329     }
330     if (cursym == endsym)
331     {
332       return;
333     }
334 }
335
336 bool parser::newConnection()
337 {
338   //EBNF: con = devicename'.'input '=' devicename['.'output]
339   bool errorOccurance = false;
340   devlink devtype = netz->finddevice(curname);
341   if (devtype != NULL)
342   {
343     connectionInName = curname;
344     smz->getsymbol(cursym, curname, curint);
345     if (cursym == dot)
346     {
347       smz->getsymbol(cursym, curname, curint);
348       devtype = netz->finddevice(connectionInName);
349       inplink ilist = netz->findinput(devtype, curname);
350       if (cursym == iosym && ilist != NULL)
351       {
352         name inputPin = curname;
353         smz->getsymbol(cursym, curname, curint);
354         if (cursym == equals) //SEARCH - you have got to here
355         {
356           smz->getsymbol(cursym, curname, curint);
357           devtype = netz->finddevice(curname);
358           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*
      scanner_mod, error* error_mod)
535 {
536   netz = network_mod;  /* make internal copies of these class pointers */
537   dmz = devices_mod;   /* so we can call functions from these classes  */
538   mmz = monitor_mod;   /* eg. to call makeconnection from the network  */
539   smz = scanner_mod;   /* class you say:                               */
540   erz = error_mod; /* netz->makeconnection(i1, i2, o1, o2, ok);   */
541   /* any other initialisation you want to do? */
542 }
```

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 written and 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
```

Listing 6: xor.gf2

2.1.2 Circuit Diagram

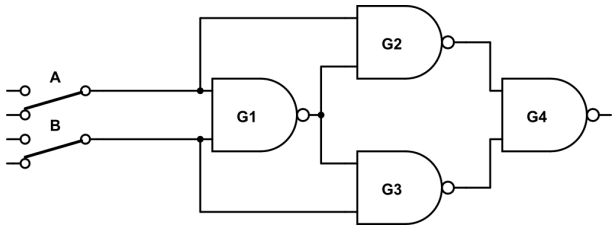


Figure 1: Circuit diagram of an XOR gate implemented using NAND gates

2.2 4-bit Adder

2.2.1 Definition File

```
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 7: 4bitadder.gf2

2.2.2 Circuit Diagram

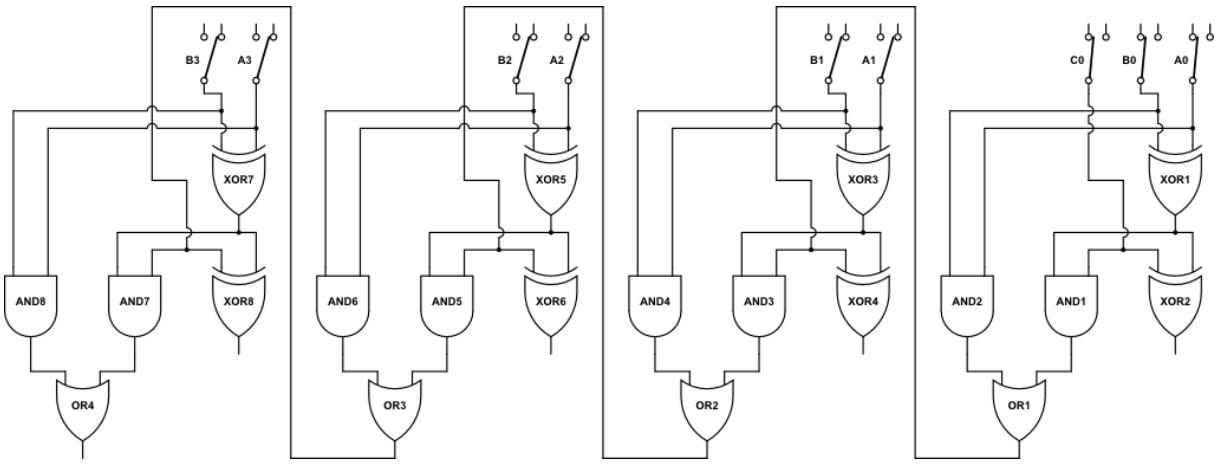


Figure 2: Circuit diagram of a 4-bit adder

2.3 Serial In Parallel Out Shift Register

2.3.1 Definition File

```
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 8: sipo.gf2

2.3.2 Circuit Diagram

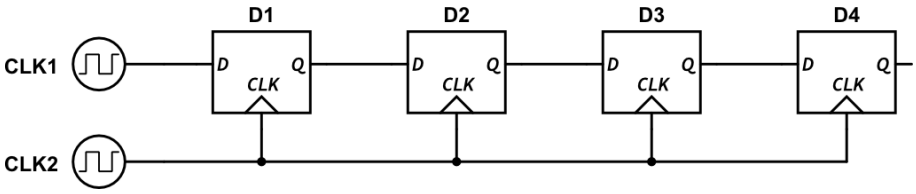


Figure 3: Circuit diagram of a serial in parallel out shift register

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

```
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 9: sipo.gf2

2.4.2 Circuit Diagram

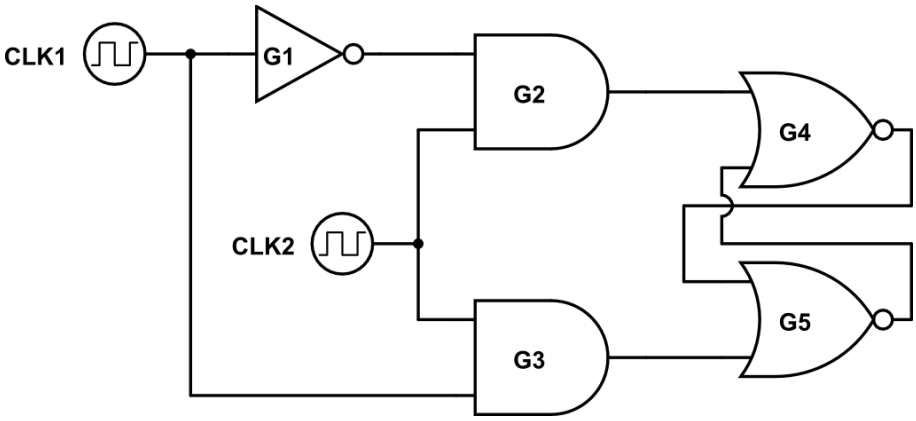
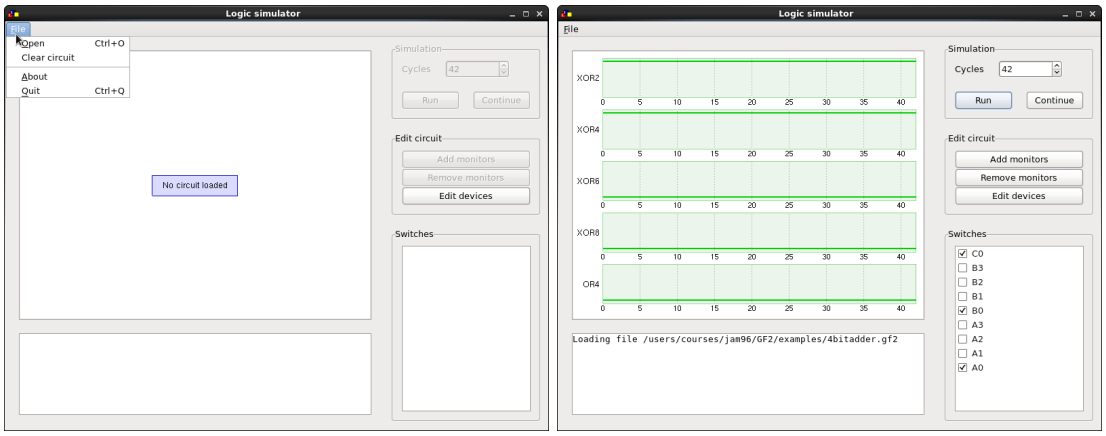


Figure 4: Circuit diagram of a Gated D Latch

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.



(a) The default view upon opening logsim (b) The view upon running a simulation

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.

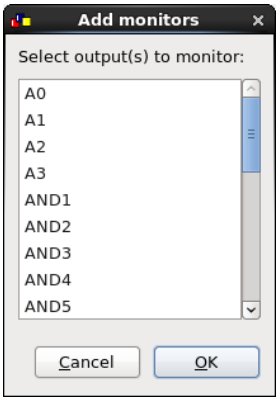


Figure 6: Add monitors dialogue

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 ouputs from a device.

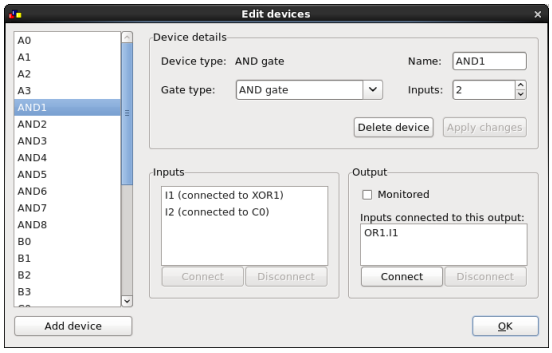


Figure 7: Edit devices dialogue