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                 */

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

class names
{

  private:
    vector<namestring> namelist;

  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>

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     namelist.push_back(str);  //Insert new string
53     return namelist.size()-1; //Return new strings internal name
54   } else {
55     return cvtname(str);
56   }
57 }
58
59 name names::cvtname (namestring str)
60 {
61   if (str == "") return blankname;
62   for (name id=0; id<namelist.size(); id++) {
63     if (namelist[id] == str) return id;   //Linear search of namelist vector
64   }
65   return blankname;
66 }
67
68 void names::writename (name id)
69 {
70   if (id == blankname) cout << "blankname";
71   else if (id > blankname && id < namelist.size()) cout << namelist[id];
72   else cout << "Incorrect id";
73 }
74
75 int names::namelength (name id)
76 {
77   if (id > blankname && id < namelist.size()) return namelist[id].length();
78   else return blankname;
79 }
80
81 namestring names::getnamestring(name id)
82 {
83   if (id > blankname && id < namelist.size()) return namelist[id];
84   else return "";
85 }
```

Listing 2: names.cc

1.2 Scanner Class

1.2.1 scanner.h

```
#ifndef scanner_h
#define scanner_h
#include <string>
#include <iostream>
#include <fstream>
#include <cstdlib>
#include "names.h"

using namespace std;

typedef int name;
typedef enum {namesym, numsym, devsym, consym, monsym, endsym, classsym, iosym, colon, semicol,
    equals, dot, badsym, eofsym} symbol;

class scanner
{
  public:
    symbol s;
    names* defnames;//Pointer to instance of names class

    scanner (names* names_mod,     //Pointer to names class
         const char* defname);  //Name of file being read
    ~scanner();            //Destructor
    void getsymbol(symbol& s,   //Symbol type read
              name& id,       //Return symbol name (if it has one)
              int& num);      //Return symbol value (if it's a number)
    void writelineerror();

  private:
    ifstream inf; //Input file
    char curch;   //Current input character
    char prevch;  //Previous input character. Used for finding line end
    bool eofile;  //True for end of file
    bool eoline;  //True for line end
    int linenum;  //Number of lines in definition file
    int cursymlen;  //Length of current symbol. Used for error printing
    string line;  //Current line contents. Used for error printing

    void getch(); //Gets next input character
    void getnumber(int& number);  //Reads number from file
    void getname(name& id); //Reads name from file
    string getline(); //Reads the line
    void skipspaces(); //Skips spaces
    void skipcomments(); //Skips comments
};

#endif
```

Listing 3: scanner.h

1.2.2 scanner.cc

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

using namespace std;

scanner::scanner(names* names_mod, const char* defname)
{
  defnames = names_mod;
  inf.open(defname);  //Open file
  if (!inf)
  {
    cout << "Error: cannot open file for reading" << endl;
```

```
  }
  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;
      }
    }
  }
}

void scanner::writelineerror()
{
  string errorptr;
```

```
92    for (int i = 0; i < ((int)line.length() - cursymlen); i++)
93    {
94      errorptr.push_back(' ');
95    }
96    errorptr.push_back('^');
97    cout << "Line " << linenum << ":" << endl;
98    cout << getline() << endl;     //Outputs current line
99    cout << errorptr << endl; //Outputs a caret at the error
100 }
101
102 void scanner::getch()
103 {
104   prevch = curch;
105   eofile = (inf.get(curch) == 0); //get next character
106   if (prevch == '\n')
107   {
108     linenum++;
109     line.clear();
110   }
111   else
112   {
113     line.push_back(prevch);
114   }
115 }
116
117 void scanner::getnumber(int& number)
118 {
119   number = 0;
120   cursymlen = 0;
121   while (isdigit(curch) && !eofile)
122   {
123     number *= 10;
124     number += (int(curch) - int('0'));
125     cursymlen++;
126     getch();
127   }
128 }
129
130 void scanner::getname(name& id)
131 {
132   namestring str;
133   cursymlen = 0;
134   while (isalnum(curch) && !eofile)
135   {
136     str.push_back(curch);
137     cursymlen++;
138     getch();
139   }
140   id = defnames->lookup(str);
141 }
142
143 void scanner::skipspaces()
144 {
145   while (isspace(curch) || curch == '\n')
146   {
147     getch();
148     if (eofile) break;
149   }
150 }
151
152 void scanner::skipcomments()
153 {
154   while (!(prevch == '*' && curch == '/'))
155   {
156     getch();
157     if (eofile) break;
158   }
159   getch(); //Get to next useful char
160 }
161
162 string scanner::getline()
163 {
164   if (s != semicol)
165   {
166     while (curch != ';' && !eofile && curch != '\n')
167     {
168       getch();
169     }
170     line.push_back(curch);
```

```
171     }
172     return line;
173 }
```

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 = 0, connectionDone = 0, monitorDone = 0;
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       devicePresent = 0;
27       deviceDone = 1;
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       connectionPresent = 0;
41       connectionDone = 1;
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       monitorPresent = 0;
55       monitorDone = 1;
56       monitorList();
57     }
58     else
59     {
60       while (cursym != devsym && cursym != consym && cursym != monsym && cursym != eofsym)
61       {
62         smz->getsymbol(cursym, curname, curint);
63       }
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    if (!devicePresent)
87    {
88      smz->getsymbol(cursym, curname, curint);
89      if (cursym == classsym)
90      {
91        newDevice(curname);
92        devicePresent = 1;
93      }
94      else if (cursym == endsym)
95      {
96        erz->newError(3); //must have at least one device
97      }
98      else
99      {
100       erz->newError(4); //need a device type
101     }
102     smz->getsymbol(cursym, curname, curint);
103   }
104   while (cursym == semicol)
105   {
106     smz->getsymbol(cursym, curname, curint);
107     if (cursym == classsym)
108     {
109       newDevice(curname);
110     }
111     else if (cursym == endsym)
112     {
113       return;
114     }
115     else if (cursym == consym | cursym == devsym | cursym == monsym)
116     {
117       erz->newError(32);//Block must be terminated with 'END'
118       return;
119     }
120     else
121     {
122       erz->newError(5);//Expecting device name or END after semicolon (device name must start with
      letter)
123     }
124     smz->getsymbol(cursym, curname, curint);
125   }
126   erz->newError(24);//must end line in semicolon
127   while (cursym != semicol && cursym != endsym && cursym != eofsym)
128   {
129     smz->getsymbol(cursym, curname, curint);
130   }
131   if (cursym == semicol)
132   {
133     deviceList();
134   }
135   if (cursym == endsym)
136   {
137     return;
138   }
139 }
140
141
142 void parser::newDevice(int deviceType)
143 {
```

```
144   //EBNF: dev = clock|switch|gate|dtype|xor
145   smz->getsymbol(cursym, curname, curint);
146   if (cursym == namesym)
147   {
148     name devName = curname;
149     if (deviceType == 10)
150     {
151       dmz->makedevice(dtype, devName, 0, correctOperation); //create DTYPE with name devName
152       return;
153     }
154     if (deviceType == 11)
155     {
156       dmz->makedevice(xorgate, devName, 2, correctOperation); //create XOR with name devName
157       return;
158     }
159     smz->getsymbol(cursym, curname, curint);
160     if (cursym == colon)
161     {
162       smz->getsymbol(cursym, curname, curint);
163       if (cursym == numsym)
164       {
165         switch (deviceType)
166         {
167           case 4:
168             if (curint > 0)
169             {
170               dmz->makedevice(aclock, devName, curint, correctOperation); //create clock with curint
      and devName
171             }
172             else
173             {
174               erz->newError(6);//clock must have number greater than 0
175             }
176             break;
177           case 5:
178             if (curint == 1 || curint == 0)
179             {
180               dmz->makedevice(aswitch, devName, curint, correctOperation);//create switch with
    curint and devName
181             }
182             else
183             {
184               erz->newError(7);//switch must have either 0 or 1
185             }
186             break;
187           case 6:
188           case 7:
189           case 8:
190           case 9:
191             if (curint > 0 && curint < 17)
192             {
193               switch (deviceType)
194               {
195                 case 6:
196                   dmz->makedevice(andgate, devName, curint, correctOperation);//create and gate with
      curint and devName
197                   break;
198                 case 7:
199                   dmz->makedevice(nandgate, devName, curint, correctOperation);//create nand gate
    with curint and devName
200                   break;
201                 case 8:
202                   dmz->makedevice(orgate, devName, curint, correctOperation);//create or gate with
    curint and devName
203                   break;
204                 case 9:
205                   dmz->makedevice(norgate, devName, curint, correctOperation);//create nor gate with
      curint and devName
206                   break;
207                 default:
208                   cout << "How on earth have you managed to get here?" << endl;
209               }
210             }
211             else
212             {
213               erz->newError(8);//must have between 1 and 16 inputs to a GATE
214             }
215             break;
216           default:
```

```
217             cout << "Please do not deduct marks if this message is displayed" << endl;
218         }
219         return;
220       }
221       else
222       {
223         erz->newError(9);//clock needs clock cycle number
224       }
225     }
226     else
227     {
228       erz->newError(10);//need colon after name for CLOCK/SWITCH/GATE type
229     }
230   }
231   else
232   {
233     erz->newError(11);//name must begin with name starting with letter and only containing letter
      number and _
234   }
235 }
236
237 void parser::connectionList()
238 {
239   //EBNF: connections = 'CONNECTIONS' {con ';'} 'END'
240   if (!connectionPresent)
241   {
242     smz->getsymbol(cursym, curname, curint);
243     if (cursym == endsym)
244     {
245       erz->newWarning(0);//No Connections
246       return;
247     }
248     else if (cursym == namesym)
249     {
250       newConnection();
251       connectionPresent = 1;
252     }
253     else
254     {
255       erz->newError(12);//connection must start with the name of a device
256     }
257     smz->getsymbol(cursym, curname, curint);
258   }
259   while (cursym == semicol)
260   {
261     smz->getsymbol(cursym, curname, curint);
262     if (cursym == namesym)
263     {
264       newConnection();
265     }
266     else if (cursym == endsym)
267     {
268       return;
269     }
270     else if (cursym == consym | cursym == devsym | cursym == monsym)
271     {
272       erz->newError(32);//Block must be terminated with 'END'
273       return;
274     }
275     else
276     {
277       erz->newError(13);//connection must start with the name of a device or end of device list must
       be terminated with END (not semicolon)
278     }
279     smz->getsymbol(cursym, curname, curint);
280   }
281   erz->newError(24);//must end line in semicolon
282   while (cursym != semicol && cursym != endsym && cursym != eofsym)
283   {
284     smz->getsymbol(cursym, curname, curint);
285   }
286   if (cursym == semicol)
287   {
288     connectionList();
289   }
290   if (cursym == endsym)
291   {
292     return;
293   }
```

```
294 }
295
296 void parser::newConnection()
297 {
298   //EBNF: con = devicename '.' input '=' devicename['.'output]
299   if (smz->defnames->namelength(curname) != 0)
300   {
301     connectionInName = curname;
302     smz->getsymbol(cursym, curname, curint);
303     if (cursym == dot)
304     {
305       smz->getsymbol(cursym, curname, curint);
306       if (cursym == iosym)
307       {
308         name inputPin = curname;
309         smz->getsymbol(cursym, curname, curint);
310         if (cursym == equals) //SEARCH - you have got to here
311         {
312           smz->getsymbol(cursym, curname, curint);
313           if (smz->defnames->namelength(curname) != 0)
314           {
315             connectionOutName = curname;
316             devlink devtype = netz->finddevice(curname);
317             switch (devtype ? devtype->kind : baddevice)
318             {
319               case 7:
320                 smz->getsymbol(cursym, curname, curint);
321                 if (cursym == dot)
322                 {
323                   smz->getsymbol(cursym, curname, curint);
324                   if (cursym == iosym)
325                   {
326                     netz->makeconnection(connectionInName, inputPin, connectionOutName, curname,
    correctOperation);
327                     return;
328                   }
329                 }
330                 else
331                 {
332                   erz->newError(14);  //Expect a dot after dtype
333                 }
334               default:
335                 netz->makeconnection(connectionInName, inputPin, connectionOutName, blankname,
    correctOperation);
336                 return;
337             }
338           }
339           else
340           {
341             erz->newError(15); //Device does not exist
342           }
343         }
344         else
345         {
346           erz->newError(16);//Must specify output to connect to input with equals sign
347         }
348       }
349       else
350       {
351         erz->newError(17);//specify valid input gate after dot
352       }
353     }
354     else
355     {
356       erz->newError(18);//need to seperate connection input with a '.' (or need to specify input)
357     }
358   }
359   else
360   {
361     erz->newError(19); //Device does not exist
362   }
363 }
364
365 void parser::monitorList()
366 {
367   //EBNF: monitors = 'MONITORS' {mon ';'} 'END'
368   if (!monitorPresent)
369   {
370     smz->getsymbol(cursym, curname, curint);
```

```
371     if (cursym == endsym)
372     {
373       erz->newWarning(1);//No Monitors
374       return;
375     }
376     else if (cursym == namesym)
377     {
378       newMonitor();
379       monitorPresent = 1;
380     }
381     else
382     {
383       erz->newError(20);//monitor must start with the name of a device
384     }
385     smz->getsymbol(cursym, curname, curint);
386   }
387   while (cursym == semicol)
388   {
389     smz->getsymbol(cursym, curname, curint);
390     if (cursym == namesym)
391     {
392       newMonitor();
393     }
394     else if (cursym == endsym)
395     {
396       return;
397     }
398     else if (cursym == consym | cursym == devsym | cursym == monsym)
399     {
400       erz->newError(32);//Block must be terminated with 'END'
401       return;
402     }
403     else
404     {
405       erz->newError(21);//monitor must start with the name of a device or end of device list must be
      terminated with END (not semicolon)
406     }
407     smz->getsymbol(cursym, curname, curint);
408   }
409   erz->newError(24);//must end line in semicolon
410   while (cursym != semicol && cursym != endsym && cursym != eofsym)
411   {
412     smz->getsymbol(cursym, curname, curint);
413   }
414   if (cursym == semicol)
415   {
416     monitorList();
417   }
418   if (cursym == endsym)
419   {
420     return;
421   }
422 }
423
424 void parser::newMonitor()
425 {
426   //EBNF: mon = devicename['.'output]
427   if (smz->defnames->namelength(curname) != 0)
428   {
429     monitorName = curname;
430     devlink devtype = netz->finddevice(curname);
431     switch (devtype ? devtype->kind : baddevice)
432     {
433       case 7:
434         smz->getsymbol(cursym, curname, curint);
435         if (cursym == dot)
436         {
437           smz->getsymbol(cursym, curname, curint);
438           if (cursym == iosym)
439           {
440             mmz->makemonitor(monitorName, curname, correctOperation);
441             return;
442           }
443         }
444         else
445         {
446           erz->newError(22);  //Expect a dot after dtype
447         }
448       default:
```

```
449             mmz->makemonitor(monitorName, blankname, correctOperation);
450             return;
451         }
452     }
453     else
454     {
455         erz->newError(23);
456     }
457 }
458
459 parser::parser(network* network_mod, devices* devices_mod, monitor* monitor_mod, scanner*
        scanner_mod, error* error_mod)
460 {
461     netz = network_mod;   /* make internal copies of these class pointers */
462     dmz = devices_mod;    /* so we can call functions from these classes  */
463     mmz = monitor_mod;    /* eg. to call makeconnection from the network  */
464     smz = scanner_mod;    /* class you say:                               */
465     erz = error_mod; /* netz->makeconnection(i1, i2, o1, o2, ok);    */
466     /* any other initialisation you want to do? */
467 }
```

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

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

2.2 4-bit Adder

2.2.1 Definition File

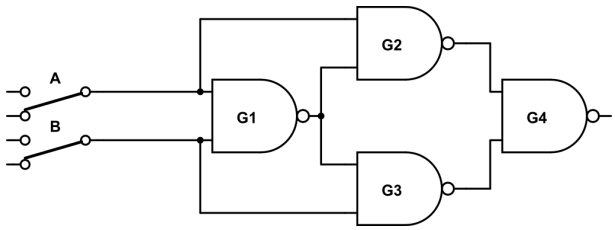

Figure 1: Circuit diagram of an XOR gate implemented using NAND gates

```
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

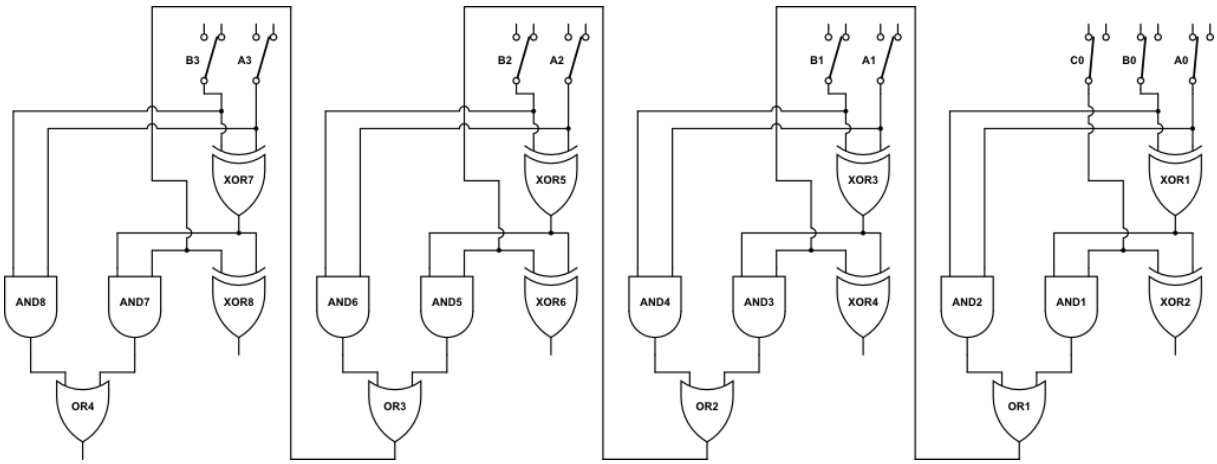

Figure 2: Circuit diagram of a 4-bit adder

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

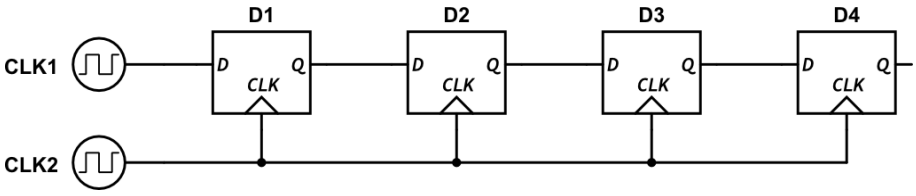

Figure 3: Circuit diagram of a serial in parallel out shift register

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

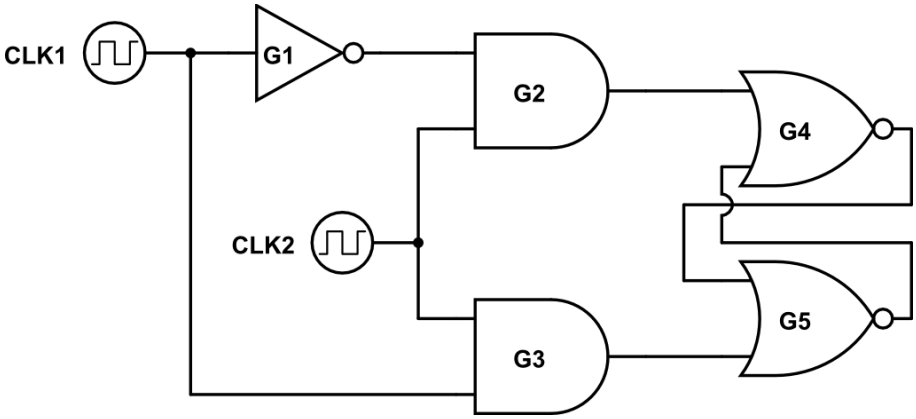

Figure 4: Circuit diagram of a Gated D Latch

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.

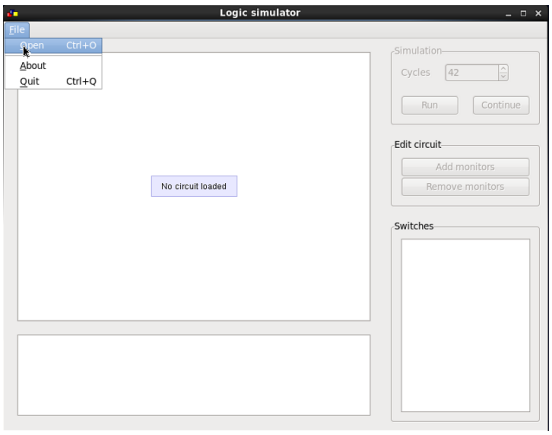

Figure 5: The default view upon opening the Logic Simulator

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.

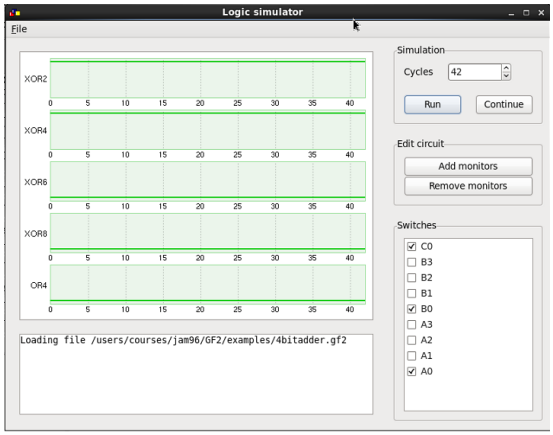

Figure 6: The view upon running a simulation

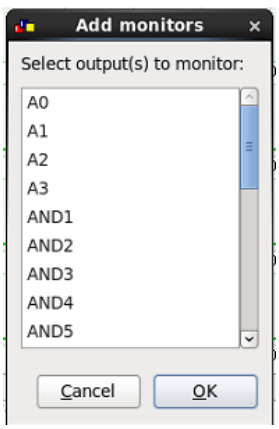

Figure 7: Add monitors dialogue

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.