

IIA GF2 Software: 2nd Interim Report

Tim Hillel (th389) - Team 8

1 Code Listings

1.1 Parser Class

1.1.1 parser.h

```
1 #ifndef parser_h
2 #define parser_h
3
4 #include "names.h"
5 #include "scanner.h"
6 #include "network.h"
7 #include "devices.h"
8 #include "monitor.h"
9 #include "error.h"
10
11 using namespace std;
12
13 class parser
14 {
15     private:
16         void deviceList();
17         void connectionList();
18         void monitorList();
19         void newDevice(int deviceType);
20         void newConnection();
21         void newMonitor();
22         network* netz; // instantiations of various classes for parser to use.
23         devices* dmz;
24         monitor* mmz;
25         scanner* smz;
26         error* erz;
27         int curint;
28         symbol cursym;
29         name curname;
30         bool correctOperation;
31         bool anyErrors;
32         int badname;
33         name monitorName;
34         name connectionInName;
35         name connectionOutName;
36         bool devicePresent;
37         bool connectionPresent;
38         bool monitorPresent;
39
40         /* put other stuff that the class uses internally here */
41         /* also declare internal functions */
42
43     public:
44         bool readin(); /* returns true if definitions file parsed ok */
45         /* Reads the definition of the logic system and builds the */
46         /* corresponding internal representation via calls to the 'Network' */
47         /* module and the 'Devices' module. */
48
49         parser(network* network_mod,
50               devices* devices_mod,
51               monitor* monitor_mod,
52               scanner* scanner_mod,
53               error* error_mod);
54         /* the constructor takes pointers to various other classes as parameters */
55 };
56
57 #endif /* parser.h */
```

Listing 1: parser.h

1.1.2 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 }
```

```

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              return;
98          }
99          else
100          {
101              erz->newError(4); //need a device type
102          }
103          smz->getsymbol(cursym, curname, curint);
104      }
105      while (cursym == semicol)
106      {
107          smz->getsymbol(cursym, curname, curint);
108          if (cursym == classsym)
109          {
110              newDevice(curname);
111          }
112          else if (cursym == endsym)
113          {
114              return;
115          }
116          else if (cursym == consym | cursym == devsym | cursym == monsym)
117          {
118              erz->newError(32); //Block must be terminated with 'END'
119              return;
120          }
121          else
122          {
123              erz->newError(5); //Expecting device name or END after semicolon (device name must start with
124              letter)
125          }
126          smz->getsymbol(cursym, curname, curint);
127      }
128      erz->newError(24); //must end line in semicolon
129      while (cursym != semicol && cursym != endsym && cursym != eofsym)
130      {
131          smz->getsymbol(cursym, curname, curint);
132      }
133      if (cursym == semicol)
134      {
135          deviceList();
136      }
137      if (cursym == endsym)
138      {
139          return;
140      }
141  }
142
143  void parser::newDevice(int deviceType)
144  {
145      //EBNF: dev = clock|switch|gate|dtype|xor
146      smz->getsymbol(cursym, curname, curint);
147      if (cursym == namesym)
148      {
149          name devName = curname;
150          if (deviceType == 10)
151          {
152              dmz->makedevice(dtype, devName, 0, correctOperation); //create DTYPE with name devName
153              return;
154          }
155          if (deviceType == 11)

```

```

156 {
157     dmz->makedevice(xorgate, devName, 2, correctOperation); //create XOR with name devName
158     return;
159 }
160 smz->getsymbol(cursym, curname, curint);
161 if (cursym == colon)
162 {
163     smz->getsymbol(cursym, curname, curint);
164     if (cursym == numsym)
165     {
166         switch (deviceType)
167         {
168             case 4:
169                 if (curint > 0)
170                 {
171                     dmz->makedevice(aclock, devName, curint, correctOperation); //create clock with curint
172                     and devName
173                 }
174                 else
175                 {
176                     erz->newError(6); //clock must have number greater than 0
177                 }
178                 break;
179             case 5:
180                 if (curint == 1 || curint == 0)
181                 {
182                     dmz->makedevice(aswitch, devName, curint, correctOperation); //create switch with
183                     curint and devName
184                 }
185                 else
186                 {
187                     erz->newError(7); //switch must have either 0 or 1
188                 }
189                 break;
190             case 6:
191             case 7:
192             case 8:
193             case 9:
194                 if (curint > 0 && curint < 17)
195                 {
196                     switch (deviceType)
197                     {
198                         case 6:
199                             dmz->makedevice(andgate, devName, curint, correctOperation); //create and gate with
200                             curint and devName
201                             break;
202                         case 7:
203                             dmz->makedevice(nandgate, devName, curint, correctOperation); //create nand gate
204                             with curint and devName
205                             break;
206                         case 8:
207                             dmz->makedevice(orgate, devName, curint, correctOperation); //create or gate with
208                             curint and devName
209                             break;
210                         case 9:
211                             dmz->makedevice(norgate, devName, curint, correctOperation); //create nor gate with
212                             curint and devName
213                             break;
214                         default:
215                             cout << "How on earth have you managed to get here?" << endl;
216                     }
217                 }
218                 else
219                 {
220                     erz->newError(8); //must have between 1 and 16 inputs to a GATE
221                 }
222                 break;
223             default:
224                 cout << "Please do not deduct marks if this message is displayed" << endl;
225         }
226     }
227     return;
228 }
else
{
    erz->newError(9); //clock needs clock cycle number
}
}
else
{

```

```

229     erz->newError(10); //need colon after name for CLOCK/SWITCH/GATE type
230 }
231 }
232 else
233 {
234     erz->newError(11); //name must begin with name starting with letter and only containing letter
        number and _
235 }
236 }
237
238 void parser::connectionList()
239 {
240     //EBNF: connections = 'CONNECTIONS' {con ';' } 'END'
241     if (!connectionPresent)
242     {
243         smz->getsymbol(cursym, curname, curint);
244         if (cursym == endsym)
245         {
246             erz->newWarning(0); //No Connections
247             return;
248         }
249         else if (cursym == namesym)
250         {
251             newConnection();
252             connectionPresent = 1;
253         }
254         else
255         {
256             erz->newError(12); //connection must start with the name of a device
257         }
258         smz->getsymbol(cursym, curname, curint);
259     }
260     while (cursym == semicol)
261     {
262         smz->getsymbol(cursym, curname, curint);
263         if (cursym == namesym)
264         {
265             newConnection();
266         }
267         else if (cursym == endsym)
268         {
269             return;
270         }
271         else if (cursym == consym | cursym == devsym | cursym == monsym)
272         {
273             erz->newError(32); //Block must be terminated with 'END'
274             return;
275         }
276         else
277         {
278             erz->newError(13); //connection must start with the name of a device or end of device list must
                be terminated with END (not semicolon)
279         }
280         smz->getsymbol(cursym, curname, curint);
281     }
282     erz->newError(24); //must end line in semicolon
283     while (cursym != semicol && cursym != endsym && cursym != eofsym)
284     {
285         smz->getsymbol(cursym, curname, curint);
286     }
287     if (cursym == semicol)
288     {
289         connectionList();
290     }
291     if (cursym == endsym)
292     {
293         return;
294     }
295 }
296
297 void parser::newConnection()
298 {
299     //EBNF: con = devicename '.' input '=' devicename['.' output]
300     if (smz->defnames->namelength(curname) != 0)
301     {
302         connectionInName = curname;
303         smz->getsymbol(cursym, curname, curint);
304         if (cursym == dot)
305         {

```

```

306 smz->getsymbol(cursym, curname, curint);
307 if (cursym == iosym)
308 {
309     name inputPin = curname;
310     smz->getsymbol(cursym, curname, curint);
311     if (cursym == equals) //SEARCH - you have got to here
312     {
313         smz->getsymbol(cursym, curname, curint);
314         if (smz->defnames->namelength(curname) != 0)
315         {
316             connectionOutName = curname;
317             devlink devtype = netz->finddevice(curname);
318             switch (devtype ? devtype->kind : baddevice)
319             {
320                 case 7:
321                     smz->getsymbol(cursym, curname, curint);
322                     if (cursym == dot)
323                     {
324                         smz->getsymbol(cursym, curname, curint);
325                         if (cursym == iosym)
326                         {
327                             netz->makeconnection(connectionInName, inputPin, connectionOutName, curname,
correctOperation);
328                             return;
329                         }
330                     }
331                     else
332                     {
333                         erz->newError(14); //Expect a dot after dtype
334                     }
335                     default:
336                         netz->makeconnection(connectionInName, inputPin, connectionOutName, blankname,
correctOperation);
337                         return;
338                     }
339                 }
340                 else
341                 {
342                     erz->newError(15); //Device does not exist
343                 }
344             }
345             else
346             {
347                 erz->newError(16); //Must specify output to connect to input with equals sign
348             }
349         }
350         else
351         {
352             erz->newError(17); //specify valid input gate after dot
353         }
354     }
355     else
356     {
357         erz->newError(18); //need to seperate connection input with a '.' (or need to specify input)
358     }
359 }
360 else
361 {
362     erz->newError(19); //Device does not exist
363 }
364 }
365
366 void parser::monitorList()
367 {
368     //EBNF: monitors = 'MONITORS' {mon ';' } 'END'
369     if (!monitorPresent)
370     {
371         smz->getsymbol(cursym, curname, curint);
372         if (cursym == endsym)
373         {
374             erz->newWarning(1); //No Monitors
375             return;
376         }
377         else if (cursym == namesym)
378         {
379             newMonitor();
380             monitorPresent = 1;
381         }
382         else

```

```

383     {
384         erz->newError(20); //monitor must start with the name of a device
385     }
386     smz->getsymbol(cursym, curname, curint);
387 }
388 while (cursym == semicol)
389 {
390     smz->getsymbol(cursym, curname, curint);
391     if (cursym == namesym)
392     {
393         newMonitor();
394     }
395     else if (cursym == endsym)
396     {
397         return;
398     }
399     else if (cursym == consym | cursym == devsym | cursym == monsym)
400     {
401         erz->newError(32); //Block must be terminated with 'END'
402         return;
403     }
404     else
405     {
406         erz->newError(21); //monitor must start with the name of a device or end of device list must be
            terminated with END (not semicolon)
407     }
408     smz->getsymbol(cursym, curname, curint);
409 }
410 erz->newError(24); //must end line in semicolon
411 while (cursym != semicol && cursym != endsym && cursym != eofsym)
412 {
413     smz->getsymbol(cursym, curname, curint);
414 }
415 if (cursym == semicol)
416 {
417     monitorList();
418 }
419 if (cursym == endsym)
420 {
421     return;
422 }
423 }
424
425 void parser::newMonitor()
426 {
427     //EBNF: mon = devicename['.'output]
428     if (smz->defnames->namelength(curname) != 0)
429     {
430         monitorName = curname;
431         devlink devtype = netz->finddevice(curname);
432         switch (devtype ? devtype->kind : baddevice)
433         {
434             case 7:
435                 smz->getsymbol(cursym, curname, curint);
436                 if (cursym == dot)
437                 {
438                     smz->getsymbol(cursym, curname, curint);
439                     if (cursym == iosym)
440                     {
441                         mmz->makemonitor(monitorName, curname, correctOperation);
442                         return;
443                     }
444                 }
445                 else
446                 {
447                     erz->newError(22); //Expect a dot after dtype
448                 }
449             default:
450                 mmz->makemonitor(monitorName, blankname, correctOperation);
451                 return;
452         }
453     }
454     else
455     {
456         erz->newError(23);
457     }
458 }
459

```

```

460 parser::parser(network* network_mod, devices* devices_mod, monitor* monitor_mod, scanner*
    scanner_mod, error* error_mod)
461 {
462     netz = network_mod; /* make internal copies of these class pointers */
463     dmz = devices_mod; /* so we can call functions from these classes */
464     mmz = monitor_mod; /* eg. to call makeconnection from the network */
465     smz = scanner_mod; /* class you say: */
466     erz = error_mod; /* netz->makeconnection(i1, i2, o1, o2, ok); */
467     /* any other initialisation you want to do? */
468 }

```

Listing 2: parser.cc

I was mainly responsible for `parser.cc`, with input from Jamie Magee on the `connectionList` and `monitorList` functions. (Final ratio was around 75% my work, 25% his work.)

1.2 Error Class

1.2.1 error.h

```

1 #ifndef error_h
2 #define error_h
3
4 #include "scanner.h"
5
6 typedef string errorstring;
7
8 class error
9 {
10     private:
11         int errorCount;
12         int warningCount;
13         vector<errorstring> errorlist;
14         vector<errorstring> warninglist;
15         scanner* smz;
16     public:
17         error(scanner* scanner_mod);
18         void newError(int errorCode);
19         void newWarning(int warningCode);
20         bool anyErrors(); //outputs total number of errors and warnings and returns 1 if any errors are
            present
21 };
22
23 #endif /* error_h */

```

Listing 3: error.h

1.2.2 error.cc

```

1 #include <iostream>
2 #include "error.h"
3
4 using namespace std;
5
6 /* The error handling class */
7
8 /* Name storage and retrieval routines */
9
10 error::error(scanner* scanner_mod) /* the constructor */
11 {
12     //Populate errorlist with reserved words
13     errorlist.push_back("Error 0x0000: Must have device list first in definition file, initialised
        with 'DEVICES'"); //0
14     errorlist.push_back("Error 0x0001: Must have connection list second (after devices and before
        monitors) in definition file, initialised by 'CONNECTIONS'"); //1
15     errorlist.push_back("Error 0x0002: Must have monitor list last (after devices list and connections
        ) in definition file, initialised by 'MONITORS'"); //2
16     errorlist.push_back("Error 0x0003: Must have at least one device in devices list"); //3
17     errorlist.push_back("Error 0x0004: Need device type for device definition"); //4
18     errorlist.push_back("Error 0x0005: Expecting device name or END after semicolon (device name must
        start with letter)"); //5 device list

```



```

19 errorlist.push_back("Error 0x0006: Clock must have integer clock number greater than 0"); //6
20 errorlist.push_back("Error 0x0007: Switch must be set to either 0 or 1"); //7
21 errorlist.push_back("Error 0x0008: Must specify an integer number of inputs between 1 and 16 to a
    GATE"); //8
22 errorlist.push_back("Error 0x0006: Clock must have integer clock number greater than 0"); //9
23 errorlist.push_back("Error 0x000A: Need colon after name for CLOCK/SWITCH/GATE type"); //10
24 errorlist.push_back("Error 0x000B: Name must start with letter and only contain letter number and
    '_ '"); //11
25 errorlist.push_back("Error 0x000C: Connection must start with the name of a device"); //12
26 errorlist.push_back("Error 0x000D: Expecting device name or END after semicolon (device name must
    start with letter)"); //13 connection list
27 errorlist.push_back("Error 0x000E: Expect a dot after DTYPE"); //14
28 errorlist.push_back("Error 0x000F: Input device called in connection list does not exist"); //15
29 errorlist.push_back("Error 0x0010: Must specify output to connect to input with equals sign "); //
    16
30 errorlist.push_back("Error 0x0011: Must specify valid input gate after dot"); //17
31 errorlist.push_back("Error 0x0012: Need to specify valid input gate separated from device name by a
    ' '"); //18
32 errorlist.push_back("Error 0x0013: Output device called in connection list does not exist"); //19
33 errorlist.push_back("Error 0x0014: Monitor must start with the name of a valid device"); //20
34 errorlist.push_back("Error 0x0015: Expecting device name or END after semicolon (device name must
    start with letter)"); //21 monitor list
35 errorlist.push_back("Error 0x0016: Expect a dot after DTYPE as must specify output to monitor in
    monitor list"); //22
36 errorlist.push_back("Error 0x0017: Bad device monitor"); //23
37 errorlist.push_back("Error 0x0018: Need semicolon at end of previous line"); //24
38 errorlist.push_back("Error 0x0019: Must only be one devices list"); //25
39 errorlist.push_back("Error 0x001A: There must be one 'DEVICES' block, it may not have been
    initialised properly"); //26
40 errorlist.push_back("Error 0x001B: Devices block must be initialised with 'DEVICES'"); //27
41 errorlist.push_back("Error 0x001C: Must only be one connections list"); //28
42 errorlist.push_back("Error 0x001D: Must only be one monitors list"); //29
43 errorlist.push_back("Error 0x001E: There must be one 'CONNECTIONS' block, it may not have been
    initialised properly"); //30
44 errorlist.push_back("Error 0x001F: There must be one 'MONITOR' block, it may not have been
    initialised properly"); //31
45 errorlist.push_back("Error 0x0020: Block must be terminated with 'END'"); //32
46 errorCount = 0;
47 warningCount = 0;
48 warninglist.push_back("Warning 0x0000: You have not specified any connections. Please check this is
    what is required"); //0
49 warninglist.push_back("Warning 0x0001: You have not specified any monitors. Please check this is
    what is required"); //1
50 smz = scanner_mod;
51 }
52
53 void error::newError(int errorCode)
54 {
55     if (errorCode >= 0 && errorCode < errorlist.size())
56     {
57         smz->writelnerror();
58         cout << errorlist[errorCode] << endl;
59         errorCount++;
60     }
61     else
62     {
63         cout << "Internal software error: Error code " << errorCode << " does not exist" << endl;
64     }
65 }
66
67 void error::newWarning(int warningCode)
68 {
69     cout << warninglist[warningCode] << endl; //don't display where warning occurs
70     warningCount++;
71 }
72
73 bool error::anyErrors()
74 {
75     if (errorCount == 0)
76     {
77         if (warningCount == 1)
78         {
79             cout << "There are no errors and 1 warning" << endl;
80         }
81         if (warningCount > 1)
82         {
83             cout << "There are no errors and " << warningCount << " warnings" << endl;
84         }
85         return 0;

```

```

86 }
87 if (errorCount == 1)
88 {
89     if (warningCount == 0)
90     {
91         cout << "There is 1 error" << endl;
92     }
93     else if (warningCount == 1)
94     {
95         cout << "There is 1 error and 1 warning" << endl;
96     }
97     else if (warningCount > 1)
98     {
99         cout << "There is 1 error and " << warningCount << "warnings" << endl;
100     }
101     return 1;
102 }
103 if (errorCount > 1)
104 {
105     if (warningCount == 0)
106     {
107         cout << "There are " << errorCount << " errors" << endl;
108     }
109     else if (warningCount == 1)
110     {
111         cout << "There are " << errorCount << " errors and 1 warning" << endl;
112     }
113     else if (warningCount > 1)
114     {
115         cout << "There are " << errorCount << " errors and " << warningCount << "warnings" << endl;
116     }
117     return 1;
118 }
119 }

```

Listing 4: error.cc

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 5: xor.gf2

2.1.2 Circuit Diagram

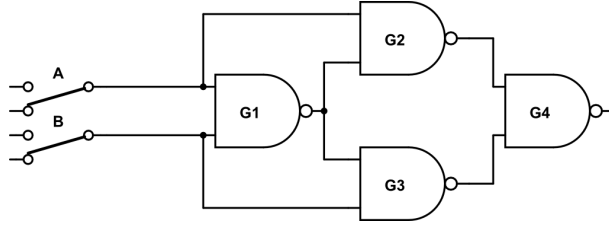


Figure 1: Circuit diagram of an XOR gate implemented using NAND gates

2.2 4-bit Adder

2.2.1 Definition File

```
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;
```

```

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 6: 4bitadder.gf2

2.2.2 Circuit Diagram

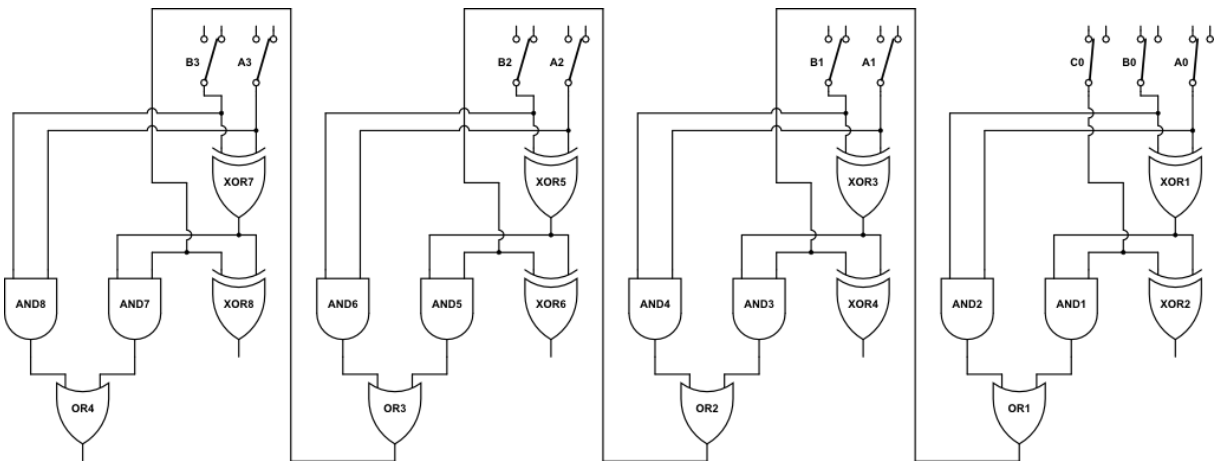


Figure 2: Circuit diagram of a 4-bit adder

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 7: sipo.gf2

2.3.2 Circuit Diagram

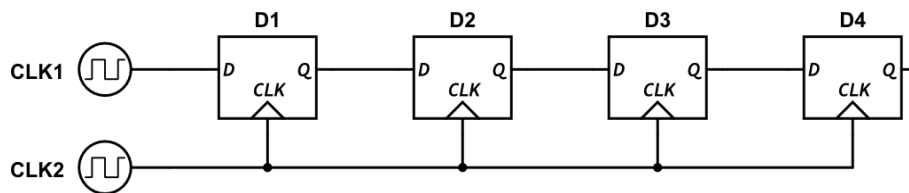


Figure 3: Circuit diagram of a serial in parallel out shift register

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 8: sipo.gf2

2.4.2 Circuit Diagram

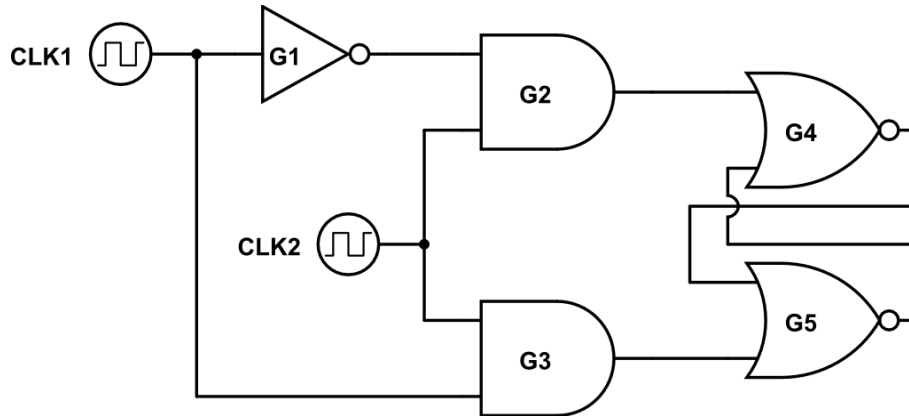


Figure 4: Circuit diagram of a Gated D Latch

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.

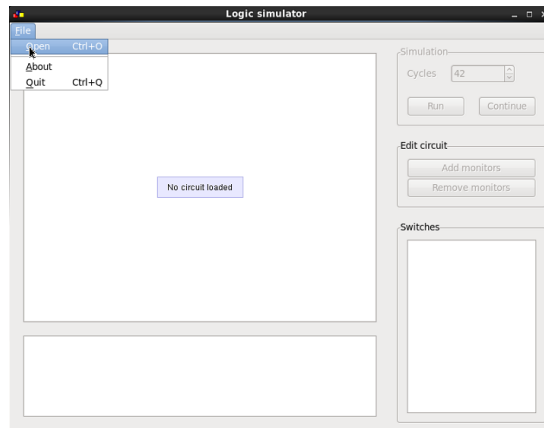


Figure 5: The default view upon opening the Logic Simulator

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.

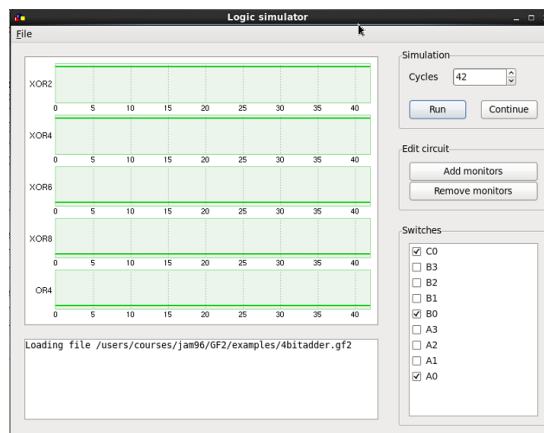


Figure 6: The view upon running a simulation

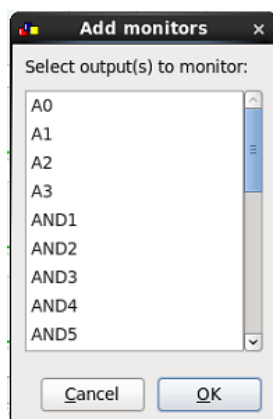


Figure 7: Add monitors dialogue

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.