

# IIA GF2 Software: 2nd Interim Report

Tim Hillel (th389) - Team 8

## 1 User Guide

### 1. Open logic simulator

#### Using terminal

Open a terminal and setting the directory to the `./logsim/src` folder. Start the application by entering `./logsim`. To open a known definition file, follow this by the directory of the definition file, and skip instruction 2.

#### Using file browser

Browse to `./logsim/src` and double click on the `logsim` executable file.

### 2. Open definition file

Click **File** menu and select **Open** to open the file selection dialogue. Only definition files with the extension `.gf2` will be displayed. The default directory is `./examples`. Browse to the required definition file and double click to open it.

### 3. Errors

If the definition file contains errors, the message **Failed to load file** will display in the display panel and any errors in the definition file will be written to the message window. If there are no errors the message **No simulation results. Use the run button.** will display, and the simulator is ready to run.

### 4. Run simulation

Enter a number of clock cycles required in the **Simulation** box, either using the up and down arrows or entering the number as text. The default value is 42. Click run once the required value has been entered.

### 5. Output

The monitored signals will be displayed in the left display panel. The simulation can be continued by the specified number of clock cycles by pressing the continue button. This can be repeated multiple times. Scrollbars will appear once there is too much data to display in the display panel; these can be used to scroll across the display.

### 6. Change switch states

Click the check box next to a switch in the **Switches** box to change its state (ticked box will be high, unticked low). Run or continue the simulation to see the effect on the circuit.

### 7. Add monitors

Click the **Add monitors** button to open the dialogue box. Select the desired output(s) to monitor and then click **OK** to add these outputs to the display panel. Multiple outputs can be selected by holding the **ctrl** button.

**8. Remove monitors** To remove monitors, click the **Remove monitors** button to open the dialogue box. Select the desired monitor(s) to remove and then click **OK**. Multiple outputs can be selected by holding the **ctrl** button. The selected monitors will be removed from the display panel.

### 9. Edit devices

Click the **Edit devices** button to open the dialogue box. Select the required device to edit in the left hand panel, and use the options within the dialogue box to change the device name, type and number of inputs (if applicable).

Note that changes made in steps 6-9 will only make changes within the simulation and will not change the definition files.

## 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 1: xor.gf2

#### 2.1.2 Circuit Diagram

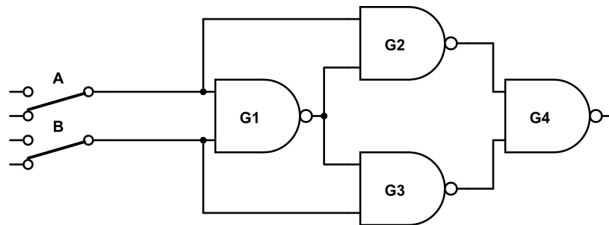


Figure 1: Circuit diagram of an XOR gate implemented using NAND gates

### 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;
57
58 XOR5.I1 = A2;
59 XOR5.I2 = B2;
60 AND5.I1 = XOR5;
61 AND5.I2 = OR2;
62 AND6.I1 = A2;
63 AND6.I2 = B2;
64 XOR6.I1 = XOR5;
65 XOR6.I2 = OR2;
66 OR3.I1 = AND5;
67 OR3.I2 = AND6;
68
69 /* MSB Adder */
70 XOR7.I1 = A3;
71 XOR7.I2 = B3;
72 AND7.I1 = XOR7;
73 AND7.I2 = OR3;
74 AND8.I1 = A3;
75 AND8.I2 = B3;
76 XOR8.I1 = XOR7;
77 XOR8.I2 = OR3;
78 OR4.I1 = AND7;
79 OR4.I2 = AND8;
80 END
81
82 MONITORS
83 /* Outputs */
84 XOR2;
85 XOR4;
86 XOR6;
87 XOR8;
88 OR4; /* Carry out */
89 END

```

Listing 2: 4bitadder.gf2

### 2.2.2 Circuit Diagram

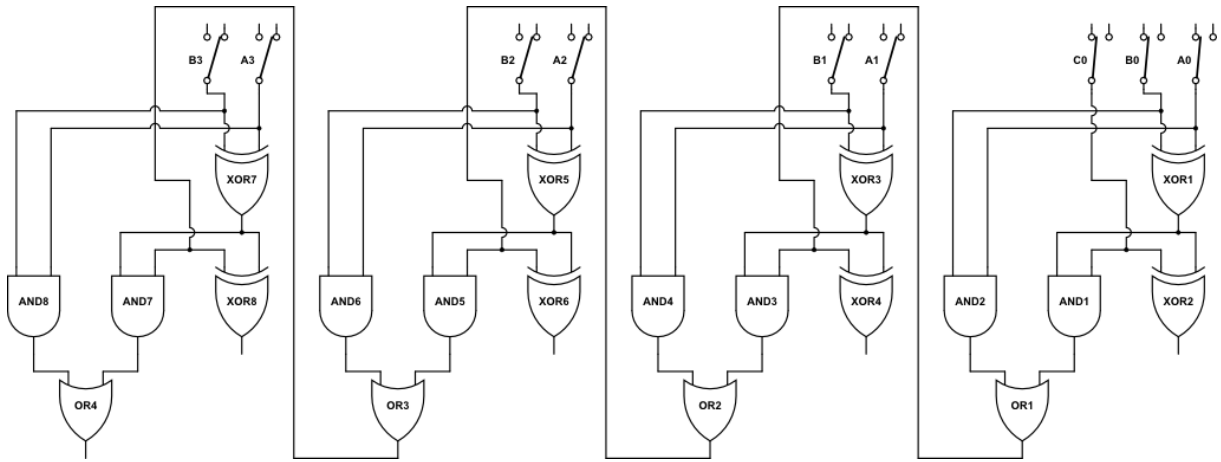


Figure 2: Circuit diagram of a 4-bit adder

## 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 3: sipo.gf2

### 2.3.2 Circuit Diagram

**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.

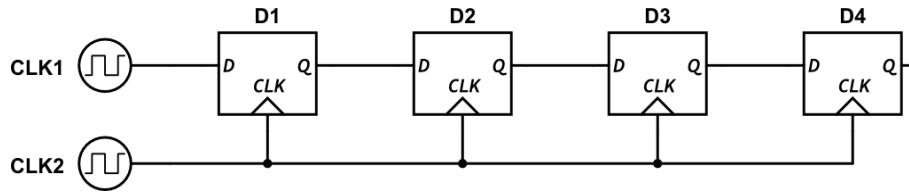


Figure 3: Circuit diagram of a serial in parallel out shift register

## 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 4: sipo.gf2

### 2.4.2 Circuit Diagram

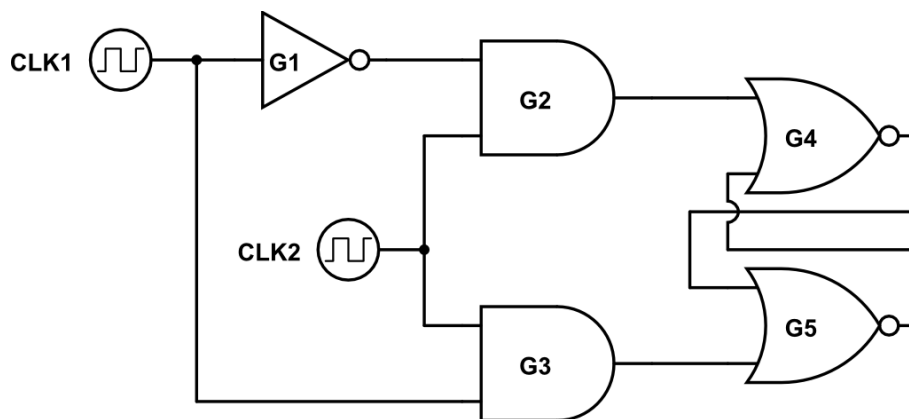


Figure 4: Circuit diagram of a Gated D Latch

**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 Code Listings

### 3.1 Parser Class

#### 3.1.1 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         bool newDevice(int deviceType);
20         bool newConnection();
21         bool 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; // integer, symbol and name returned by scanner
28         symbol cursym;
29         name curname;
30         bool correctOperation; // bools to check for errors
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 5: parser.h

#### 3.1.2 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 = false, connectionDone = false, monitorDone = false;
13     cursym = badsym;
14     while (cursym != eofsym)
15     {
16         if (cursym != devsym && cursym != consym && cursym != monsym)
17         {
18             smz->getsymbol(cursym, curname, curint);
19         }
20         if (cursym == devsym)
21         {
22             if (deviceDone)
23             {
24                 erz->newError(25); //Must only be one devices list
25             }
26             deviceDone = true;
27             deviceList();
28         }
29         else if (cursym == consym)
30         {
31             if (!deviceDone)
32             {
33                 erz->newError(0); //must have device list first
34             }
35             if (connectionDone)
36             {
37                 erz->newError(28); //Must only be one connections list
38             }
39             connectionDone = true;
40             connectionList();
41         }
42         else if (cursym == monsym)
43         {
44             if (!deviceDone | !connectionDone)
45             {
46                 erz->newError(2); //Must have monitor list last
47             }
48             if (monitorDone)
49             {
50                 erz->newError(29); //Must only be one Monitors list
51             }
52             monitorDone = true;
53             monitorList();
54         }
55     }
56     else if (cursym != eofsym)
57     {
58         while (cursym != devsym && cursym != consym && cursym != monsym && cursym != eofsym)
59         {
60             smz->getsymbol(cursym, curname, curint);
61             erz->countSymbols();
62         }
63         erz->symbolError(deviceDone, connectionDone, monitorDone);
64     }
65 }
66 if (!deviceDone)
67 {
68     erz->newError(26); //There must be a DEVICES block, it may not have been initialised properly
69 }
70 if (!connectionDone)
71 {
72     erz->newError(30); //There must be a CONNECTIONS block, it may not have been initialised properly
73 }
74 if (!monitorDone)
75 {
76     erz->newError(31); //There must be a MONITORS block, it may not have been initialised properly
77 }
78 netz->checknetwork(correctOperation);
79 anyErrors = erz->anyErrors();
80 return (correctOperation && !anyErrors);
81 }
82
83 void parser::deviceList()
84 {

```

```

85 //EBNF: devices = 'DEVICES' dev {';' dev} ';' 'END'
86 bool deviceError;
87 if (!devicePresent)
88 {
89     smz->getsymbol(cursym, curname, curint);
90     if (cursym == classsym)
91     {
92         deviceError = newDevice(curname);
93         devicePresent = true;
94     }
95     else if (cursym == endsym)
96     {
97         erz->newError(3); //must have at least one device
98         return;
99     }
100    else
101    {
102        erz->newError(4); //need a device type
103    }
104    if (!deviceError)
105    {
106        smz->getsymbol(cursym, curname, curint);
107    }
108 }
109 while (cursym == semicol)
110 {
111     smz->getsymbol(cursym, curname, curint);
112     if (cursym == classsym)
113     {
114         deviceError = newDevice(curname);
115     }
116     else if (cursym == endsym)
117     {
118         return;
119     }
120     else if (cursym == consym | cursym == devsym | cursym == monsym)
121     {
122         erz->newError(32); //Block must be terminated with 'END'
123         return;
124     }
125     else
126     {
127         erz->newError(5); //Expecting device name or END after semicolon (device name must start with
128         letter)
129     }
130     if (!deviceError)
131     {
132         smz->getsymbol(cursym, curname, curint);
133     }
134 }
135 if (!deviceError) erz->newError(24); //must end line in semicolon
136 while (cursym != semicol && cursym != endsym && cursym != eofsym)
137 {
138     smz->getsymbol(cursym, curname, curint);
139 }
140 if (cursym == semicol)
141 {
142     deviceList();
143 }
144 if (cursym == endsym)
145 {
146     return;
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
183                     curint and devName
184                 }
185                 else
186                 {
187                     erz->newError(6); //clock must have number greater than 0
188                     errorOccurance=true;
189                 }
190                 break;
191             case 5:
192                 if (curint == 1 || curint == 0)
193                 {
194                     dmz->makedevice(aswitch, devName, curint, correctOperation); //create switch with
195                     curint and devName
196                 }
197                 else
198                 {
199                     erz->newError(7); //switch must have either 0 or 1
200                     errorOccurance=true;
201                 }
202                 break;
203             case 6:
204             case 7:
205             case 8:
206             case 9:
207                 if (curint > 0 && curint < 17)
208                 {
209                     switch (deviceType)
210                     {
211                         case 6:
212                             dmz->makedevice(andgate, devName, curint, correctOperation); //create and gate
213                             with curint and devName
214                             break;
215                         case 7:
216                             dmz->makedevice(nandgate, devName, curint, correctOperation); //create nand gate
217                             with curint and devName
218                             break;
219                         case 8:
220                             dmz->makedevice(orgate, devName, curint, correctOperation); //create or gate with
221                             curint and devName
222                             break;
223                         case 9:
224                             dmz->makedevice(norgate, devName, curint, correctOperation); //create nor gate
225                             with curint and devName
226                             break;
227                         default:
228                             cout << "How on earth have you managed to get here?" << endl;
229                     }
230                 }
231                 else
232                 {
233                     erz->newError(8); //must have between 1 and 16 inputs to a GATE
234                     errorOccurance=true;
235                 }
236                 break;
237             default:
238                 cout << "Please do not deduct marks if this message is displayed" << endl;
239             }
240         }
241     }
242     return errorOccurance;
243 }

```

```

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 ilst = netz->findinput(devtype, curname);
350             if (cursym == iosym && ilst != 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                                 }
375                                 else
376                                 {
377                                     erz->newError(34); //Not valid output for dtype
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 6: parser.cc

I was mainly responsible for `parser.cc`, with input from Jamie Magee on the `connectionList` and `monitorList` functions. (Final ratio was around 80% my work, 20% his work.)

## 3.2 Error Class

### 3.2.1 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     int symbolCount;
14     bool firstTime;
15     vector<errorstring> errorlist;
16     vector<errorstring> warninglist;
17     scanner* smz;
18 public:
19     error(scanner* scanner_mod);
20     void newError(int errorCode);
21     void symbolError(bool deviceDone, bool connectionDone, bool monitorDone);
22     void newWarning(int warningCode);
23     void countSymbols();
24
25     bool anyErrors(); //outputs total number of errors and warnings and returns 1 if any errors are
        present
26 };
27
28 #endif /* error_h */
```

Listing 7: error.h

### 3.2.2 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: Device already exists with this name, please choose an
    alternative name"); //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 'MONITORS' block, it may not have been
    initialised properly"); //31
45 errorlist.push_back("Error 0x0020: Block must be terminated with 'END'"); //32
46 errorlist.push_back("Error 0x0021: Cannot name a device as a reserved word, for a list of reserved
    words check reservedWords.txt in docs"); //33
47 errorlist.push_back("Error 0x0022: Not a valid output for a dtype"); //34
48 errorlist.push_back("RESERVED"); //35 RESERVED FOR symbolError() function
49
50 errorCount = 0;
51 warningCount = 0;
52 symbolCount = 0;
53 firstTime=true;
54 warninglist.push_back("Warning 0x0000: You have not specified any connections. Please check this
    is what is required"); //0
55 warninglist.push_back("Warning 0x0001: You have not specified any monitors. Please check this is
    what is required"); //1
56 smz = scanner_mod;
57 }
58
59 void error::newError(int errorCode)
60 {
61     if (errorCode >= 0 && errorCode < errorlist.size())
62     {
63         smz->writelineerror();
64         cout << errorlist[errorCode] << endl;
65         errorCount ++;
66     }
67     else
68     {
69         cout << "Internal software error: Error code " << errorCode << " does not exist" << endl;
70     }
71 }
72
73 void error::newWarning(int warningCode)
74 {
75     cout << warninglist[warningCode] << endl; //don't display where warning occurs
76     warningCount ++;
77 }
78
79 void error::countSymbols()
80 {
81     if(firstTime)
82     {
83         symbolCount=0;
84     }
85     symbolCount++;
86     firstTime=false;
87 }
88
89 void error::symbolError(bool deviceDone, bool connectionDone, bool monitorDone)
90 {
91     smz->writelineerror();

```

```

92     cout << "Error 0x0023: There are " << symbolCount << " unexpected symbols before this line." <<
93     endl;
94     if (!deviceDone)
95     {
96         cout << "Expected DEVICES block" << endl;
97     }
98     else if (!connectionDone)
99     {
100         cout << "Expected CONNECTIONS block" << endl;
101     }
102     else if (!monitorDone)
103     {
104         cout << "Expected MONITORS block" << endl;
105     }
106     errorCount ++;
107 }
108
109 bool error::anyErrors()
110 {
111     if (errorCount == 0)
112     {
113         if (warningCount == 1)
114         {
115             cout << "There are no errors and 1 warning" << endl;
116         }
117         if (warningCount > 1)
118         {
119             cout << "There are no errors and " << warningCount << " warnings" << endl;
120         }
121         return 0;
122     }
123     if (errorCount == 1)
124     {
125         if (warningCount == 0)
126         {
127             cout << "There is 1 error" << endl;
128         }
129         else if (warningCount == 1)
130         {
131             cout << "There is 1 error and 1 warning" << endl;
132         }
133         else if (warningCount > 1)
134         {
135             cout << "There is 1 error and " << warningCount << " warnings" << endl;
136         }
137         return 1;
138     }
139     if (errorCount > 1)
140     {
141         if (warningCount == 0)
142         {
143             cout << "There are " << errorCount << " errors" << endl;
144         }
145         else if (warningCount == 1)
146         {
147             cout << "There are " << errorCount << " errors and 1 warning" << endl;
148         }
149         else if (warningCount > 1)
150         {
151             cout << "There are " << errorCount << " errors and " << warningCount << " warnings" << endl;
152         }
153         return 1;
154     }
155 }

```

Listing 8: error.cc