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 definiton 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. Mulitple outputs can be selected by holding the `ctrl` button.

8. Remove monitors To remove monitors, clock the `Remove monitors` button to open the dialogue box. Select the desired monitor(s) to remove and then click `OK`. Mulitple 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

```
DEVICES
SWITCH S1:0;
SWITCH S2:1;
NAND G1:2;
NAND G2:2;
NAND G3:2;
NAND G4:2;
END

CONNECTIONS
G1.I1 = S1;
G1.I2 = S2;
G2.I1 = S1;
G2.I2 = G1;
G3.I1 = G1;
G3.I2 = S2;
G4.I1 = G2;
G4.I2 = G3;
END

MONITORS
S1;
S2;
G4;
END
```

Listing 1: xor.gf2

2.1.2 Circuit Diagram

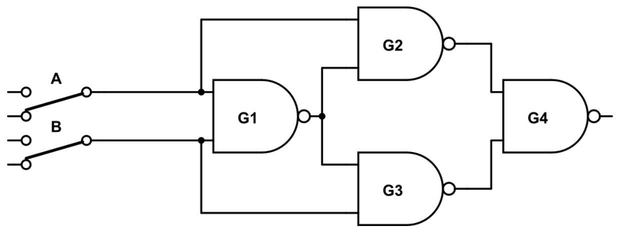

Figure 1: Circuit diagram of an XOR gate implemented using NAND gates

2.2 4-bit Adder

2.2.1 Definition File

```
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 2: 4bitadder.gf2

2.2.2 Circuit Diagram

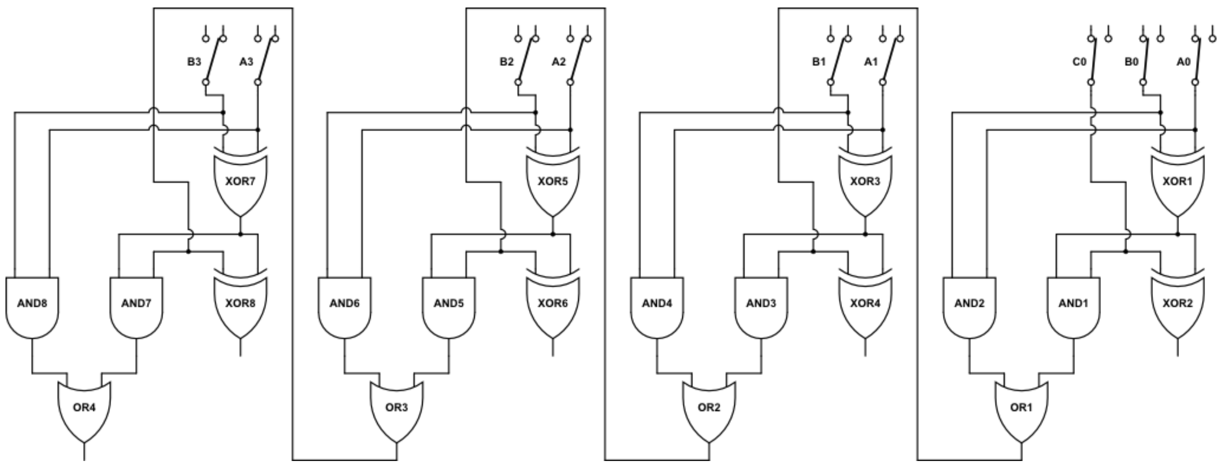

Figure 2: Circuit diagram of a 4-bit adder

2.3 Serial In Parallel Out Shift Register

2.3.1 Definition File

```
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 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.

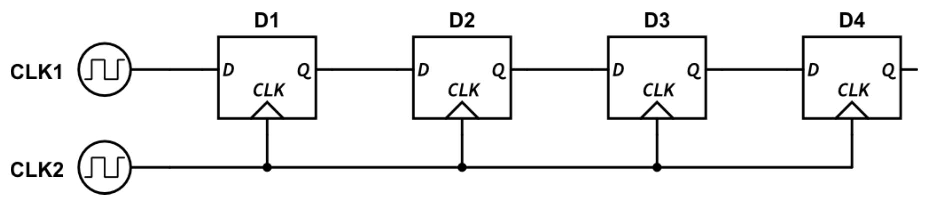

Figure 3: Circuit diagram of a serial in parallel out shift register

2.4 Gated D Latch

2.4.1 Definition File

```
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 4: sipo.gf2

2.4.2 Circuit Diagram

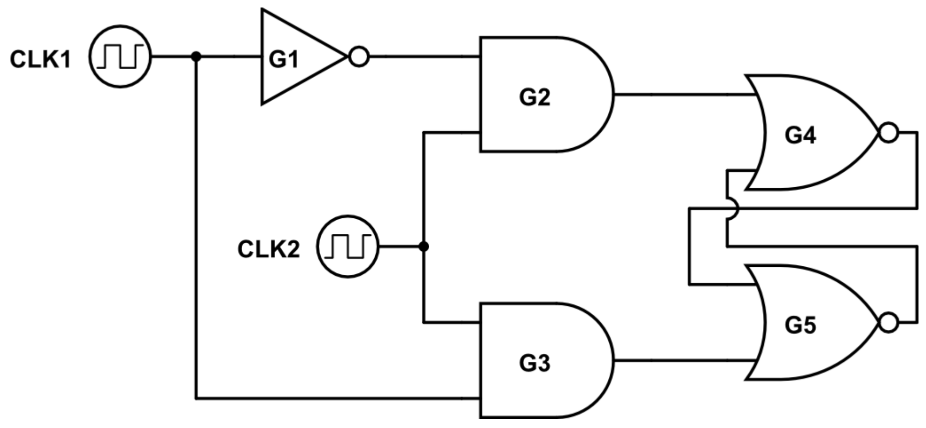

Figure 4: Circuit diagram of a Gated D Latch

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

```cpp
#ifndef parser_h
#define parser_h

#include "names.h"
#include "scanner.h"
#include "network.h"
#include "devices.h"
#include "monitor.h"
#include "error.h"

using namespace std;

class parser
{
  private:
    void deviceList();
    void connectionList();
    void monitorList();
    bool newDevice(int deviceType);
    bool newConnection();
    bool newMonitor();
    network* netz; // instantiations of various classes for parser to use.
    devices* dmz;
    monitor* mmz;
    scanner* smz;
    error* erz;
    int curint;  //integer, symbol and name returned by scanner
    symbol cursym;
    name curname;
    bool correctOperation; //bools to check for errors
    bool anyErrors;
    int badname;
    name monitorName;
    name connectionInName;
    name connectionOutName;
    bool devicePresent;
    bool connectionPresent;
    bool monitorPresent;

    /* put other stuff that the class uses internally here */
    /* also declare internal functions                     */

  public:
    bool readin();  /* returns true if definitions file parsed ok */
    /* Reads the definition of the logic system and builds the             */
    /* corresponding internal representation via calls to the 'Network'   */
    /* module and the 'Devices' module.                                    */

    parser(network* network_mod,
         devices* devices_mod,
         monitor* monitor_mod,
         scanner* scanner_mod,
         error* error_mod);
    /* the constructor takes pointers to various other classes as parameters */
};

#endif /* parser_h */
```

Listing 5: parser.h

3.1.2 parser.cc

```cpp
#include <iostream>
#include "parser.h"
#include "error.h"

using namespace std;
```

```

/* The parser for the circuit definition files */

bool parser::readin(void)
{
  //EBNF: specfile = devices connections monitors
  bool deviceDone = false, connectionDone = false, monitorDone = false;
  cursym = badsym;
  while (cursym != eofsym)
  {
    if (cursym != devsym && cursym != consym && cursym != monsym)
    {
      smz->getsymbol(cursym, curname, curint);
    }
    if (cursym == devsym)
    {
      if (deviceDone)
      {
        erz->newError(25);//Must only be one devices list
      }
      deviceDone = true;
      deviceList();
    }
    else if (cursym == consym)
    {
      if (!deviceDone)
      {
        erz->newError(0); //must have device list first
      }
      if (connectionDone)
      {
        erz->newError(28);//Must only be one connections list

      }
      connectionDone = true;
      connectionList();
    }
    else if (cursym == monsym)
    {
      if (!deviceDone | !connectionDone)
      {
        erz->newError(2); //Must have monitor list last
      }
      if (monitorDone)
      {
        erz->newError(29);//Must only be one Monitors list
      }
      monitorDone = true;
      monitorList();
    }
    else if (cursym != eofsym)
    {
      while (cursym != devsym && cursym != consym && cursym != monsym && cursym != eofsym)
      {
        smz->getsymbol(cursym, curname, curint);
        erz->countSymbols();
      }
      erz->symbolError(deviceDone, connectionDone, monitorDone);
    }
  }
  if (!deviceDone)
  {
    erz->newError(26);//There must be a DEVICES block, it may not have been initialised properly
  }
  if (!connectionDone)
  {
    erz->newError(30);//There must be a CONNECTIONS block, it may not have been initialised properly
  }
  if (!monitorDone)
  {
    erz->newError(31);//There must be a MONITORS block, it may not have been initialised properly
  }
  netz->checknetwork(correctOperation);
  anyErrors = erz->anyErrors();
  return (correctOperation && !anyErrors);
}

void parser::deviceList()
{
  //EBNF: devices = 'DEVICES' dev {';' dev} ';' 'END'
  bool deviceError;
  if (!devicePresent)
  {
    smz->getsymbol(cursym, curname, curint);
    if (cursym == classsym)
    {
      deviceError = newDevice(curname);
      devicePresent = true;
    }
    else if (cursym == endsym)
    {
      erz->newError(3); //must have at least one device
      return;
    }
    else
    {
      erz->newError(4); //need a device type
    }
    if (!deviceError)
    {
      smz->getsymbol(cursym, curname, curint);
    }
  }
  while (cursym == semicol)
  {
    smz->getsymbol(cursym, curname, curint);
    if (cursym == classsym)
    {
      deviceError = newDevice(curname);
    }
    else if (cursym == endsym)
    {
      return;
    }
    else if (cursym == consym | cursym == devsym | cursym == monsym)
    {
      erz->newError(32);//Block must be terminated with 'END'
      return;
    }
    else
    {
      erz->newError(5);//Expecting device name or END after semicolon (device name must start with
    letter)
    }
    if (!deviceError)
    {
      smz->getsymbol(cursym, curname, curint);
    }
  }
  if (!deviceError) erz->newError(24);//must end line in semicolon
  while (cursym != semicol && cursym != endsym && cursym != eofsym)
  {
    smz->getsymbol(cursym, curname, curint);
  }
  if (cursym == semicol)
  {
    deviceList();
  }
  if (cursym == endsym)
  {
    return;
  }
}


bool parser::newDevice(int deviceType)
{
  //EBNF: dev = clock|switch|gate|dtype|xor
  bool errorOccurance = false;
  smz->getsymbol(cursym, curname, curint);
  if (cursym == namesym)
  {
    devlink nameCheck = netz->finddevice(curname);
    if(nameCheck==NULL)
    {
      name devName = curname;
      if (deviceType == 10)
      {
```

```
      dmz->makedevice(dtype, devName, 0, correctOperation); //create DTYPE with name devName
      return errorOccurance;
    }
    if (deviceType == 11)
    {
      dmz->makedevice(xorgate, devName, 2, correctOperation); //create XOR with name devName
      return errorOccurance;
    }
    smz->getsymbol(cursym, curname, curint);
    if (cursym == colon)
    {
      smz->getsymbol(cursym, curname, curint);
      if (cursym == numsym)
      {
        switch (deviceType)
        {
          case 4:
            if (curint > 0)
            {
              dmz->makedevice(aclock, devName, curint, correctOperation); //create clock with curint and devName
            }
            else
            {
              erz->newError(6);//clock must have number greater than 0
              errorOccurance=true;
            }
            break;
          case 5:
            if (curint == 1 || curint == 0)
            {
              dmz->makedevice(aswitch, devName, curint, correctOperation);//create switch with curint and devName
            }
            else
            {
              erz->newError(7);//switch must have either 0 or 1
              errorOccurance=true;
            }
            break;
          case 6:
          case 7:
          case 8:
          case 9:
            if (curint > 0 && curint < 17)
            {
              switch (deviceType)
              {
                case 6:
                  dmz->makedevice(andgate, devName, curint, correctOperation);//create and gate with curint and devName
                  break;
                case 7:
                  dmz->makedevice(nandgate, devName, curint, correctOperation);//create nand gate with curint and devName
                  break;
                case 8:
                  dmz->makedevice(orgate, devName, curint, correctOperation);//create or gate with curint and devName
                  break;
                case 9:
                  dmz->makedevice(norgate, devName, curint, correctOperation);//create nor gate with curint and devName
                  break;
                default:
                  cout << "How on earth have you managed to get here?" << endl;
              }
            }
            else
            {
              erz->newError(8);//must have between 1 and 16 inputs to a GATE
              errorOccurance=true;
            }
            break;
          default:
            cout << "Please do not deduct marks if this message is displayed" << endl;
        }
        return errorOccurance;
      }
```

```
        else
        {
          erz->newError(9);//clock needs clock cycle number
          errorOccurance=true;
        }
      }
      else
      {
        erz->newError(10);//need colon after name for CLOCK/SWITCH/GATE type
        errorOccurance=true;
      }
    }
    else
    {
      erz->newError(34);//attempting to give two devices the same name, choose an alternative name
      errorOccurance=true;
    }
  }
  else if (cursym!=badsym)
  {
    erz->newError(33);//using reserved word as device name
    errorOccurance=true;
  }
  else
  {
    erz->newError(11);//name must begin with letter and only containing letter number and _
    errorOccurance=true;
  }
  return errorOccurance;
}

void parser::connectionList()
{
  //EBNF: connections = 'CONNECTIONS' {con ';'} 'END'
  bool connectionError;
  if (!connectionPresent)
  {
    smz->getsymbol(cursym, curname, curint);
    if (cursym == endsym)
    {
      if (!connectionPresent)
      {
        erz->newWarning(0);//No Connections
      }
      return;
    }
    else if (cursym == namesym)
    {
      connectionError = newConnection();
      connectionPresent = true;
    }
    else
    {
      erz->newError(12);//connection must start with the name of a device
    }
    if (!connectionError)
    {
      smz->getsymbol(cursym, curname, curint);
    }
  }
  while (cursym == semicol)
  {
    smz->getsymbol(cursym, curname, curint);
    if (cursym == namesym)
    {
      connectionError = newConnection();
    }
    else if (cursym == endsym)
    {
      return;
    }
    else if (cursym == consym | cursym == devsym | cursym == monsym)
    {
      erz->newError(32);//Block must be terminated with 'END'
      return;
    }
    else
    {
      erz->newError(13);//connection must start with the name of a device or end of device list must
     be terminated with END (not semicolon)
    }
    if (!connectionError)
    {
      smz->getsymbol(cursym, curname, curint);
    }
  }
  if (!connectionError) erz->newError(24);//must end line in semicolon
  while (cursym != semicol && cursym != endsym && cursym != eofsym)
  {
    smz->getsymbol(cursym, curname, curint);
  }
  if (cursym == semicol)
  {
    connectionList();
  }
  if (cursym == endsym)
  {
    return;
  }
}

bool parser::newConnection()
{
  //EBNF: con = devicename'.'input '=' devicename['.'output]
  bool errorOccurance = false;
  devlink devtype = netz->finddevice(curname);
  if (devtype != NULL)
  {
    connectionInName = curname;
    smz->getsymbol(cursym, curname, curint);
    if (cursym == dot)
    {
      smz->getsymbol(cursym, curname, curint);
      devtype = netz->finddevice(connectionInName);
      inplink ilist = netz->findinput(devtype, curname);
      if (cursym == iosym && ilist != NULL)
      {
        name inputPin = curname;
        smz->getsymbol(cursym, curname, curint);
        if (cursym == equals) //SEARCH - you have got to here
        {
          smz->getsymbol(cursym, curname, curint);
          devtype = netz->finddevice(curname);
          if (devtype != NULL)
          {
            connectionOutName = curname;
            switch (devtype ? devtype->kind : baddevice)
            {
              case 7:
                smz->getsymbol(cursym, curname, curint);
                if (cursym == dot)
                {
                  smz->getsymbol(cursym, curname, curint);
                  outplink olist = netz->findoutput(devtype, curname);
                  if (cursym == iosym && olist != NULL)
                  {
                    netz->makeconnection(connectionInName, inputPin, connectionOutName, curname,
    correctOperation);
                    return errorOccurance;
                  }
                  else
                  {
                    erz->newError(34); //Not valid output for dtype
                  }
                }
                else
                {
                  erz->newError(14);  //Expect a dot after dtype
                  errorOccurance=true;
                }
              default:
                netz->makeconnection(connectionInName, inputPin, connectionOutName, blankname,
    correctOperation);
                return errorOccurance;
            }
          }
          else
          {
            erz->newError(15); //Device does not exist
            errorOccurance=true;
          }
        }
        else
        {
          erz->newError(16);//Must specify output to connect to input with equals sign
          errorOccurance=true;
        }
      }
      else
      {
        erz->newError(17);//specify valid input gate after dot
        errorOccurance=true;
      }
    }
    else
    {
      erz->newError(18);//need to seperate connection input with a '.' (or need to specify input)
      errorOccurance=true;
    }
  }
  else
  {
    erz->newError(19); //Device does not exist
    errorOccurance=true;
  }
  return errorOccurance;
}

void parser::monitorList()
{
  //EBNF: monitors = 'MONITORS' {mon ';'} 'END'
  bool monitorError;
  if (!monitorPresent)
  {
    smz->getsymbol(cursym, curname, curint);
    if (cursym == endsym)
    {
      if (!monitorPresent)
      {
        erz->newWarning(1);//No Monitors
      }
      return;
    }
    else if (cursym == namesym)
    {
      monitorError = newMonitor();
      monitorPresent = true;
    }
    else
    {
      erz->newError(20);//monitor must start with the name of a device
    }
    if (!monitorError)
    {
      smz->getsymbol(cursym, curname, curint);
    }
  }
  while (cursym == semicol)
  {
    smz->getsymbol(cursym, curname, curint);
    if (cursym == namesym)
    {
      monitorError = newMonitor();
    }
    else if (cursym == endsym)
    {
      return;
    }
    else if (cursym == consym | cursym == devsym | cursym == monsym)
    {
      erz->newError(32);//Block must be terminated with 'END'
      return;
    }
    else
    {
      erz->newError(21);//monitor must start with the name of a device or end of device list must be terminated with END (not semicolon)
    }
    if (!monitorError)
    {
      smz->getsymbol(cursym, curname, curint);
    }
  }
  if (!monitorError) erz->newError(24);//must end line in semicolon
  while (cursym != semicol && cursym != endsym && cursym != eofsym)
  {
    smz->getsymbol(cursym, curname, curint);
  }
  if (cursym == semicol)
  {
    monitorList();
  }
  if (cursym == endsym)
  {
    return;
  }
}

bool parser::newMonitor()
{
  //EBNF: mon = devicename['.'output]
  bool errorOccurance = false;
  devlink devtype = netz->finddevice(curname);
  if (devtype != NULL)
  {
    monitorName = curname;
    switch (devtype ? devtype->kind : baddevice)
    {
      case 7:
        smz->getsymbol(cursym, curname, curint);
        if (cursym == dot)
        {
          smz->getsymbol(cursym, curname, curint);
          outplink olist = netz->findoutput(devtype, curname);
          if (cursym == iosym && olist != NULL)
          {
            mmz->makemonitor(monitorName, curname, correctOperation);
            return errorOccurance;
          }
          else
          {
            erz->newError(34); //Not valid output for dtype
          }
        }
        else
        {
          erz->newError(22);  //Expect a dot after dtype
          errorOccurance=true;
        }
      default:
        mmz->makemonitor(monitorName, blankname, correctOperation);
        return errorOccurance;
    }
  }
  else
  {
    erz->newError(23);
    errorOccurance=true;
  }
  return errorOccurance;
}

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

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

```
#ifndef error_h
#define error_h

#include "scanner.h"

typedef string errorstring;

class error
{
  private:
    int errorCount;
    int warningCount;
    int symbolCount;
    bool firstTime;
    vector<errorstring> errorlist;
    vector<errorstring> warninglist;
    scanner* smz;
  public:
    error(scanner* scanner_mod);
    void newError(int errorCode);
    void symbolError(bool deviceDone, bool connectionDone, bool monitorDone);
    void newWarning(int warningCode);
    void countSymbols();

    bool anyErrors();//outputs total number of errors and warnings and returns 1 if any errors are
    present
};

#endif /* error_h */
```

Listing 7: error.h

3.2.2 error.cc

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

using namespace std;

/* The error handling class */

/* Name storage and retrieval routines */

error::error(scanner* scanner_mod)  /* the constructor */
{
  //Populate errorlist with reserved words
  errorlist.push_back("Error 0x0000: Must have device list first in definition file, initialised
    with 'DEVICES'"); //0
  errorlist.push_back("Error 0x0001: Must have connection list second (after devices and before
    monitors) in definition file, initialised by 'CONNECTIONS'"); //1
  errorlist.push_back("Error 0x0002: Must have monitor list last (after devices list and connections
    ) in definition file, initialised by 'MONITORS'"); //2
  errorlist.push_back("Error 0x0003: Must have at least one device in devices list"); //3
  errorlist.push_back("Error 0x0004: Need device type for device definition"); //4
  errorlist.push_back("Error 0x0005: Expecting device name or END after semicolon (device name must
    start with letter)"); //5 device list
  errorlist.push_back("Error 0x0006: Clock must have integer clock number greater than 0"); //6
  errorlist.push_back("Error 0x0007: Switch must be set to either 0 or 1"); //7
  errorlist.push_back("Error 0x0008: Must specify an integer number of inputs between 1 and 16 to a
    GATE"); //8
  errorlist.push_back("Error 0x0006: Clock must have integer clock number greater than 0"); //9
  errorlist.push_back("Error 0x000A: Need colon after name for CLOCK/SWITCH/GATE type"); //10
  errorlist.push_back("Error 0x000B: Name must start with letter and only contain letter number and
    '_'"); //11
  errorlist.push_back("Error 0x000C: Connection must start with the name of a device"); //12
  errorlist.push_back("Error 0x000D: Expecting device name or END after semicolon (device name must
    start with letter)"); //13 connection list
  errorlist.push_back("Error 0x000E: Expect a dot after DTYPE"); //14
  errorlist.push_back("Error 0x000F: Input device called in connection list does not exist"); //15
  errorlist.push_back("Error 0x0010: Must specify output to connect to input with equals sign "); //
    16
  errorlist.push_back("Error 0x0011: Must specify valid input gate after dot"); //17
  errorlist.push_back("Error 0x0012: Need to specify valid input gate separated from device name by
    a '.'"); //18
  errorlist.push_back("Error 0x0013: Output device called in connection list does not exist"); //19
  errorlist.push_back("Error 0x0014: Monitor must start with the name of a valid device"); //20
  errorlist.push_back("Error 0x0015: Expecting device name or END after semicolon (device name must
    start with letter)"); //21 monitor list
  errorlist.push_back("Error 0x0016: Expect a dot after DTYPE as must specify output to monitor in
    monitor list"); //22
  errorlist.push_back("Error 0x0017: Bad device monitor"); //23
  errorlist.push_back("Error 0x0018: Need semicolon at end of previous line"); //24
  errorlist.push_back("Error 0x0019: Must only be one devices list"); //25
  errorlist.push_back("Error 0x001A: There must be one 'DEVICES' block, it may not have been
    initialised properly");//26
  errorlist.push_back("Error 0x001B: Device already exists with this name, please choose an
    alternative name"); //27
  errorlist.push_back("Error 0x001C: Must only be one connections list"); //28
  errorlist.push_back("Error 0x001D: Must only be one monitors list"); //29
  errorlist.push_back("Error 0x001E: There must be one 'CONNECTIONS' block, it may not have been
    initialised properly");//30
  errorlist.push_back("Error 0x001F: There must be one 'MONITORS' block, it may not have been
    initialised properly");//31
  errorlist.push_back("Error 0x0020: Block must be terminated with 'END'");//32
  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
  errorlist.push_back("Error 0x0022: Not a valid output for a dtype");//34
  errorlist.push_back("RESERVED");//35 RESERVED FOR symbolError() function

  errorCount = 0;
  warningCount = 0;
  symbolCount = 0;
  firstTime=true;
  warninglist.push_back("Warning 0x0000: You have not specified any connections. Please check this
    is what is required");//0
  warninglist.push_back("Warning 0x0001: You have not specified any monitors. Please check this is
    what is required");//1
  smz = scanner_mod;
}

void error::newError(int errorCode)
{
  if (errorCode >= 0 && errorCode < errorlist.size())
  {
    smz->writelineerror();
    cout << errorlist[errorCode] << endl;
    errorCount ++;
  }
  else
  {
    cout << "Internal software error: Error code " << errorCode << " does not exist" << endl;
  }
}

void error::newWarning(int warningCode)
{
  cout << warninglist[warningCode] << endl; //don't display where warning occurs
  warningCount ++;
}

void error::countSymbols()
{
  if(firstTime)
  {
    symbolCount=0;
  }
  symbolCount++;
  firstTime=false;
}

void error::symbolError(bool deviceDone, bool connectionDone, bool monitorDone)
{
    smz->writelineerror();
    cout << "Error 0x0023: There are " << symbolCount <<" unexpected symbols before this line." << endl;
    if (!deviceDone)
    {
      cout << "Expected DEVICES block" << endl;
    }
    else if (!connectionDone)
    {
      cout << "Expected CONNECTIONS block" << endl;
    }
    else if (!monitorDone)
    {
      cout << "Expected MONITORS block" << endl;
    }
    errorCount ++;
}

bool error::anyErrors()
{
  if (errorCount == 0)
  {
    if (warningCount == 1)
    {
      cout << "There are no errors and 1 warning" << endl;
    }
    if (warningCount > 1)
    {
      cout << "There are no errors and " << warningCount << " warnings" << endl;
    }
    return 0;
  }
  if (errorCount == 1)
  {
    if (warningCount == 0)
    {
      cout << "There is 1 error" << endl;
    }
    else if (warningCount == 1)
    {
      cout << "There is 1 error and 1 warning" << endl;
    }
    else if (warningCount > 1)
    {
      cout << "There is 1 error and " << warningCount << "warnings" << endl;
    }
    return 1;
  }
  if (errorCount > 1)
  {
    if (warningCount == 0)
    {
      cout << "There are " << errorCount << " errors" << endl;
    }
    else if (warningCount == 1)
    {
      cout << "There are " << errorCount << " errors and 1 warning" << endl;
    }
    else if (warningCount > 1)
    {
      cout << "There are " << errorCount << " errors and " << warningCount << "warnings" << endl;
    }
    return 1;
  }
}
```

Listing 8: error.cc