IIA GF2 Software: Final Report

Jamie Magee (jam96)
Team 8
Gonville & Caius College

Contents

1 Description

Our logic simulator is able to simulate any number of circuits which include the following devices:

- Clocks
- Switches
- AND gates (Up to 16 inputs)
- NAND gates (Up to 16 inputs)
- OR gates (Up to 16 inputs)
- NOR gates (Up to 16 inputs)
- XOR gates
- D-Type flip-flops
- Signal generators

Figure 1: UML Class diagram of our logic simulator

2 Development Style

We split the development of our logic simulator into five major phases: specification, design, implementation, testing and maintenance. The timeframe was then decided for each task and each task was assigned to either a team member or the whole team, depending on the nature of the task.

Each member of the team was also assigned a general project role as follows:

Project manager: (T Hillel) - Responsible for project planning including delegation of tasks and ensuring that the project runs to the set timescale.

Programming administrator: (J Magee) - Responsible for upkeep of the project directory including performing builds and keeping legacy versions of the simulator.

Client representative: (M Jackson) - Responsible for ensuring that the project meets the client's requirements for the logic simulator as defined in Appendix A of the GF2 Project Handout.

We made significant use of git for revision control, as well as GitHub for tracking bugs in the software and features required by the client.

3 My Contribution

I took on responsibility for writing the names and scanner classes. In addition, I wrote approximately 25% of the parser class. Once the majority of the software was written, I designed a definition file for each error and warning our logic simulator can throw and then wrote a shell script which would attempt to run each definition file, and record the output from our logic simulator. The shell script can be found in code listing 6

4 Testing

We used two main tests of testing - unit and system testing - both of which are industry standard practices. For our unit testing, Martin wrote an errors class which compared the actual output from various units of code, to the expected output. For system testing I wrote a shell script which passed definition files to the logic simulator and recorded the output in a text file. There were two variatons on the shell script: One which ran known good definition files and therefore had to input the commands to run the simulation in addition to recording the output; Another which ran known bad definition files and only expected parsing errors which it recorded.

5 Conclusions

A Code Listings

A.1 Names Class

A.1.1 names.h

```
#ifndef names_h
#define names_h

#include <string>
#include <vector>

using namespace std;

//const int maxnames  = 200;  /* max number of distinct names */
//const int maxlength = 8;    /* max chars in a name string   */
const int blankname = -1;   /* special name                 */

const int lastreservedname = 33;

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

class names
{

  private:
    vector<namestring> namelist; //Stores a list of reserved and declared names

  public:
    name lookup(namestring str);
    /* Returns the internal representation of the name given in character  */
    /* form.  If the name is not already in the name table, it is          */
    /* automatically inserted.                                             */

    name cvtname(namestring str);
    /* Returns the internal representation of the name given in character  */
    /* form.  If the name is not in the name table then 'blankname' is     */
    /* returned.                                                           */

    void writename(name id);
    /* Prints out the given name on the console                            */

    int namelength(name id);
    /* Returns length ie number of characters in given name                */

    namestring getnamestring(name id);
    /* Returns the namestring for the given name                           */

    names(void);
    /* names initialises the name table.  This procedure is called at      */
    /* system initialisation before any of the above procedures/functions  */
    /* are used.                                                           */
};

#endif /* names_h */
```

Listing 1: names.h

A.1.2 names.cc

```
#include "names.h"
#include <iostream>
#include <string>
#include <cstdlib>

using namespace std;

/* Name storage and retrieval routines */
```

```

names::names(void)  /* the constructor */
{
  //Populate namelist with reserved words
  namelist.push_back("DEVICES"); //0
  namelist.push_back("CONNECTIONS"); //1
  namelist.push_back("MONITORS"); //2
  namelist.push_back("END"); //3
  namelist.push_back("CLOCK"); //4
  namelist.push_back("SWITCH"); //5
  namelist.push_back("AND"); //6
  namelist.push_back("NAND"); //7
  namelist.push_back("OR"); //8
  namelist.push_back("NOR"); //9
  namelist.push_back("DTYPE"); //10
  namelist.push_back("XOR"); //11
  namelist.push_back("SIGGEN"); //12
  namelist.push_back("I1"); //13
  namelist.push_back("I2"); //14
  namelist.push_back("I3"); //15
  namelist.push_back("I4"); //16
  namelist.push_back("I5"); //17
  namelist.push_back("I6"); //18
  namelist.push_back("I7"); //19
  namelist.push_back("I8"); //20
  namelist.push_back("I9"); //21
  namelist.push_back("I10"); //22
  namelist.push_back("I11"); //23
  namelist.push_back("I12"); //24
  namelist.push_back("I13"); //25
  namelist.push_back("I14"); //26
  namelist.push_back("I15"); //27
  namelist.push_back("I16"); //28
  namelist.push_back("DATA"); //29
  namelist.push_back("CLK"); //30
  namelist.push_back("SET"); //31
  namelist.push_back("CLEAR"); //32
  namelist.push_back("Q"); //33
  namelist.push_back("QBAR"); //34
}

name names::lookup(namestring str)
{
  if (cvtname(str) == blankname)
  {
    namelist.push_back(str);  //Insert new string
    return namelist.size() - 1; //Return new strings internal name
  }
  else
  {
    return cvtname(str);
  }
}

name names::cvtname(namestring str)
{
  if (str == "") return blankname;
  for (name id = 0; id < namelist.size(); id++)
  {
    if (namelist[id] == str) return id;   //Linear search of namelist vector
  }
  return blankname;
}

void names::writename(name id)
{
  if (id == blankname) cout << "blankname";
  else if (id > blankname && id < namelist.size()) cout << namelist[id];
  else cout << "Incorrect id";
}

int names::namelength(name id)
{
  if (id > blankname && id < namelist.size()) return namelist[id].length();
  else return blankname;
}
```

```

namestring names::getnamestring(name id)
{
  if (id > blankname && id < namelist.size()) return namelist[id];
  else return "";
}
```

Listing 2: names.cc

A.2 Scanner Class

A.2.1 scanner.h

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

using namespace std;

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

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

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

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

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

#endif
```

Listing 3: scanner.h

A.2.2 scanner.cc

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

```

using namespace std;

scanner::scanner(names* names_mod, const char* defname, bool& ok)
{
  nmz = names_mod;
  ok = 1;
  inf.open(defname);  //Open file
  if (!inf)
  {
    cout << "Error: cannot open file for reading" << endl;
    ok = 0;
  }
  eofile = (inf.get(curch) == 0); //Get first character
  linenum = 1;
  s = badsym;//in case getline is called before getsymbol
}

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

void scanner::getsymbol(symbol& s, name& id, int& num, string& numstring)
{
  s = badsym;
  cursymlen = 0;
  skipspaces();
  if (eofile) s = eofsym;
  else
  {
    if (isdigit(curch))
    {
      s = numsym;
      getnumber(num, numstring);
    }
    else
    {
      if (isalpha(curch))
      {
        getname(id);
        if (id == 0) s = devsym;
        else if (id == 1) s = consym;
        else if (id == 2) s = monsym;
        else if (id == 3) s = endsym;
        else if (id > 3 && id < 13) s = classsym;
        else if (id > 12 && id < 35) s = iosym;
        else s = namesym;
      }
      else
      {
        switch (curch)
        {
          case '=':
            s = equals;
            getch();
            break;
          case ';':
            s = semicol;
            getch();
            break;
          case ':':
            s = colon;
            getch();
            break;
          case '.':
            s = dot;
            getch();
            break;
          case '/':
            getch();
            if (curch == '*')
            {
              getch();
              skipcomments();
              getsymbol(s, id, num, numstring);
            }
            break;
          default:
            s = badsym;
            getch();
            break;
        }
        cursymlen = 1;
      }
    }
  }
}

void scanner::writelineerror()
{
  string errorptr;
  for (int i = 0; i < ((int)line.length() - cursymlen); i++)
  {
    errorptr.push_back(' ');
  }
  errorptr.push_back('^');
  cout << "Line " << linenum << ":" << endl;
  cout << getline() << endl;    //Outputs current line
  cout << errorptr << endl; //Outputs a caret at the error
}

void scanner::getch()
{
  prevch = curch;
  eofile = (inf.get(curch) == 0); //get next character
  if (prevch == '\n') //If eoline, clear the currently stored line
  {
    linenum++;
    line.clear();
  }
  else if (prevch != '\r') //If we're not at the end of a line, add the char to the line string
  {
    line.push_back(prevch);
  }
}

void scanner::getnumber(int& number, string& numstring)
{
  numstring = "";
  number = 0;
  cursymlen = 0;
  while (isdigit(curch) && !eofile)
  {
    numstring.push_back(curch);
    number *= 10;
    number += (int(curch) - int('0'));
    cursymlen++;
    getch();
  }
}

void scanner::getname(name& id)
{
  namestring str;
  cursymlen = 0;
  while (isalnum(curch) && !eofile)
  {
    str.push_back(curch);
    cursymlen++;
    getch();
  }
  id = nmz->lookup(str);
}

void scanner::skipspaces()
{
  while (isspace(curch) || curch == '\n')
  {
    getch();
    if (eofile) break;
  }
}

void scanner::skipcomments()
{
  while (!(prevch == '*' && curch == '/'))
  {
    getch();
    if (eofile)
    {
      cout << "Reached end of file before comment was terminated" << endl;
      break;
    }
  }
  getch(); //Get to next useful char
}

string scanner::getline()
{
  if (s != semicol)
  {
    while (curch != ';' && !eofile && curch != '\n')
    {
      getch();
    }
    if (curch != '\n' && curch != '\r' && !eofile)
    {
      line.push_back(curch);
    }
  }
  return line;
}
```

Listing 4: scanner.cc

A.3 Parser Class

A.3.1 parser.cc

```
#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;
  devicePresent = connectionPresent = monitorPresent = false;
  cursym = badsym;
  while (cursym != eofsym)
  {
    if (cursym != devsym && cursym != consym && cursym != monsym)
    {
      smz->getsymbol(cursym, curname, curint, numstring);
    }
    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, numstring);
        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, numstring);
    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, numstring);
    }
  }
  while (cursym == semicol)
  {
    smz->getsymbol(cursym, curname, curint, numstring);
    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, numstring);
    }
  }
  if (!deviceError) erz->newError(24);//must end line in semicolon
  while (cursym != semicol && cursym != endsym && cursym != eofsym)
  {
    smz->getsymbol(cursym, curname, curint, numstring);
  }
  if (cursym == semicol)
  {
    deviceList();
  }
  if (cursym == endsym)
  {
    return;
  }
}


bool parser::newDevice(int deviceType)
{
  //EBNF: dev = clock|switch|gate|dtype|xor|siggen
  bool errorOccurance = false;
  smz->getsymbol(cursym, curname, curint, numstring);
  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, numstring);
      if (cursym == colon)
      {
        smz->getsymbol(cursym, curname, curint, numstring);
        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;
            case 12:
              if (isBinary(numstring))
              {
                sequence waveform;
                for (int i=0; i<numstring.length(); i++)
                {
                  waveform.push_back(numstring[i]=='1');
                }
                dmz->makesiggen(devName, waveform); //create SIGGEN with name devName
              }
              else
              {
                erz->newError(36); //Must be a binary input
                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(27);//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, numstring);
    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, numstring);
    }
  }
  while (cursym == semicol)
  {
    smz->getsymbol(cursym, curname, curint, numstring);
    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, numstring);
    }
  }
  if (!connectionError) erz->newError(24);//must end line in semicolon
  while (cursym != semicol && cursym != endsym && cursym != eofsym)
  {
    smz->getsymbol(cursym, curname, curint, numstring);
  }
  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, numstring);
    if (cursym == dot)
    {
      smz->getsymbol(cursym, curname, curint, numstring);
      devtype = netz->finddevice(connectionInName);
      inplink ilist = netz->findinput(devtype, curname);
      if (cursym == iosym && ilist != NULL)
      {
        name inputPin = curname;
        smz->getsymbol(cursym, curname, curint, numstring);
        if (cursym == equals) //SEARCH - you have got to here
        {
          smz->getsymbol(cursym, curname, curint, numstring);
          devtype = netz->finddevice(curname);
          if (devtype != NULL)
          {
            connectionOutName = curname;
            switch (devtype ? devtype->kind : baddevice)
            {
              case 7:
                smz->getsymbol(cursym, curname, curint, numstring);
                if (cursym == dot)
                {
                  smz->getsymbol(cursym, curname, curint, numstring);
                  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:
              //check the connection is unique
                if (ilist->connect==NULL)
                {
                  netz->makeconnection(connectionInName, inputPin, connectionOutName, blankname, correctOperation);
                  return errorOccurance;
                }
                else if (ilist->connect==netz->findoutput(devtype, blankname))
                {
                  namestring repeatedInput = smz->nmz->getnamestring(connectionInName);
                  namestring repeatedOutput = smz->nmz->getnamestring(connectionOutName);
                  erz->connectionWarning(repeatedInput, repeatedOutput);
                }
                else
                {
                  erz->newError(37);//attempting to input 2 ouputs into same input
                }
            }
          }
          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, numstring);
    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, numstring);
    }
  }
  while (cursym == semicol)
  {
    smz->getsymbol(cursym, curname, curint, numstring);
    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, numstring);
    }
  }
  if (!monitorError) erz->newError(24);//must end line in semicolon
  while (cursym != semicol && cursym != endsym && cursym != eofsym)
  {
    smz->getsymbol(cursym, curname, curint, numstring);
  }
  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, numstring);
        if (cursym == dot)
        {
          smz->getsymbol(cursym, curname, curint, numstring);
          outplink olist = netz->findoutput(devtype, curname);
          bool alreadyMonitored = mmz->IsMonitored(olist);
          if (!alreadyMonitored)
          {
            if (cursym == iosym && olist != NULL)
            {
              mmz->makemonitor(monitorName, curname, correctOperation);
              return errorOccurance;
            }
            else
            {
              erz->newError(34); //Not valid output for dtype
            }
          }
          else
          {
            namestring repeatedMonitor = smz->nmz->getnamestring(curname);
            erz->monitorWarning(repeatedMonitor); //repeated monitors
            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:
        outplink olist = netz->findoutput(devtype, blankname);
        bool alreadyMonitored = mmz->IsMonitored(olist);
          if (!alreadyMonitored)
          {
            mmz->makemonitor(monitorName, blankname, correctOperation);
          }
          else
          {
            namestring repeatedMonitor = smz->nmz->getnamestring(curname);
            erz->monitorWarning(repeatedMonitor); //repeated monitors
            mmz->makemonitor(monitorName, curname, correctOperation);
          }
        return errorOccurance;
    }
  }
  else
  {
    erz->newError(23);//bad device monitor
    errorOccurance=true;
  }
  return errorOccurance;
}

bool parser::isBinary(string numstring)
{
  for (int i=0; i<numstring.length(); i++)
  {
    if (numstring[i]!='0' && numstring[i]!='1')
      {
        return false;
      }
  }
  return true;
}

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 5: parser.cc

`parser.cc` was written with joint effort between myself and Tim Hillel, with Tim contributing approximately 75% of the code.

A.4 Test Scripts

A.4.1 test.sh

```bash
#!/bin/bash
echo "" > error.txt
for f in *.gf2
do
  echo "processing $f..."
  echo "processing $f..." >> test.txt
  echo -e "r 50\r\nq\r\n" | ../src/logsim $f >> test.txt
  echo -e '\n' >> test.txt
done
```

Listing 6: test.sh

A.4.2 error.sh

```bash
#!/bin/bash
echo "" > error.txt
for f in *.gf2
do
  echo "processing $f..."
  echo "processing $f..." >> error.txt
  echo ../../src/logsim $f >> error.txt
  echo -e '\n' >> error.txt
done
```

Listing 7: error.sh

B Test Definition Files

All supplied definition files and circuit diagrams were written and designed by myself.

B.1 XOR Gate

B.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 8: xor.gf2

B.1.2 Circuit Diagram

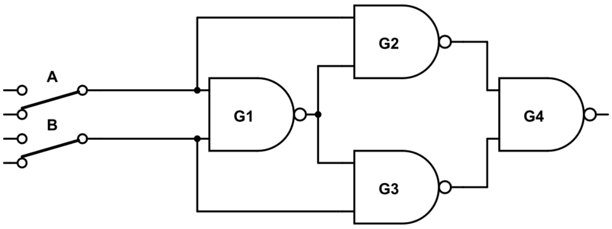

Figure 2: Circuit diagram of an XOR gate implemented using NAND gates

B.2 4-bit Adder

B.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 9: 4bitadder.gf2

B.2.2 Circuit Diagram

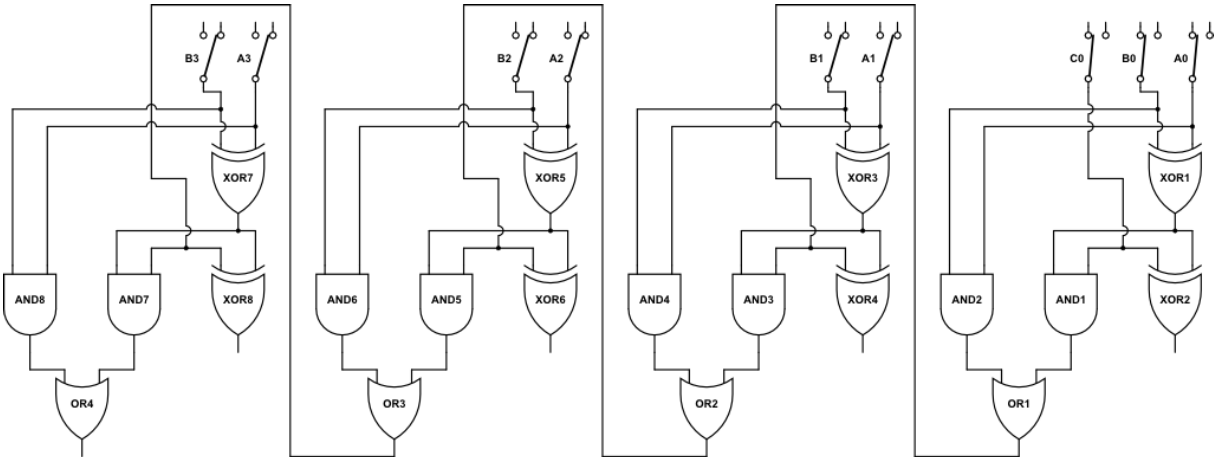

Figure 3: Circuit diagram of a 4-bit adder

B.3 Serial In Parallel Out Shift Register

B.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 10: sipo.gf2

B.3.2 Circuit Diagram

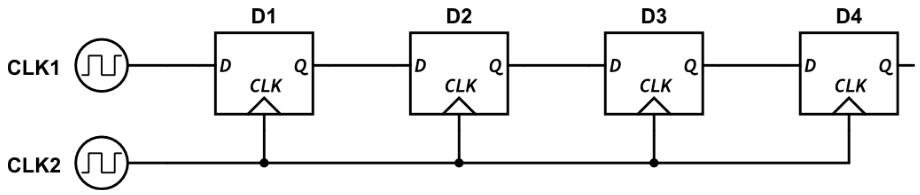

Figure 4: Circuit diagram of a serial in parallel out shift register

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. 4 was the closest achievable.

B.4 Gated D Latch

B.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 11: sipo.gf2

B.4.2 Circuit Diagram

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. 5.

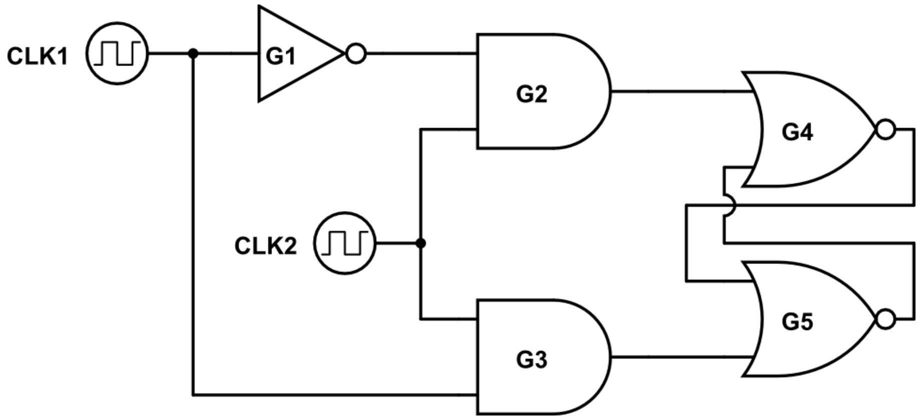

Figure 5: Circuit diagram of a Gated D Latch

C EBNF

```
specfile = devices connections monitors

devices = 'DEVICES' dev ';' {dev ';'} 'END'
connections = 'CONNECTIONS' {con ';'} 'END'
monitors = 'MONITORS' {mon ';'} 'END'

dev = clock|switch|gate|dtype|xor|siggen
con = devicename'.'input '=' devicename['.'output]
mon = devicename['.'output]

devicename = letter {'_'|letter|digit}
input = 'I'('1'|'2'|'3'|'4'|'5'|'6'|'7'|'8'|'9'|
   '10'|'11'|'12'|'13'|'14'|'15'|'16')|'DATA'|'CLK'|'SET'|'CLEAR'
output = 'Q'['BAR']

clock = 'CLOCK' devicename':'digit{digit}
switch = 'SWITCH' devicename':'('0'|'1')
gate = ('AND'|'NAND'|'OR'|'NOR') devicename':'('1'|'2'|'3'|'4'|'5'|'6'|'7'|'8'|
   '9'|'10'|'11'|'12'|'13'|'14'|'15'|'16')
dtype = 'DTYPE' devicename
xor = 'XOR' devicename
siggen = 'SIGGEN' devicename':'('0'|'1'){'0'|'1'}
```

Listing 12: EBNF

D User Guide

E File Listing