# IIA GF2 Software: 2nd Interim Report

Jamie Magee (jam96) - Team 8

## 1 Code Listings

### 1.1 Names Class

#### 1.1.1 names.h

```
#ifndef names_h
#define names_h

#include <string>
#include <vector>

using namespace std;

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

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

class names
{

  private:
    vector<namestring> namelist;

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

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

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

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

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

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

#endif /* names_h */
```

Listing 1: names.h

#### 1.1.2 names.cc

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

```
#include <cstdlib>

using namespace std;

/* 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("I1"); //12
  namelist.push_back("I2"); //13
  namelist.push_back("I3"); //14
  namelist.push_back("I4"); //15
  namelist.push_back("I5"); //16
  namelist.push_back("I6"); //17
  namelist.push_back("I7"); //18
  namelist.push_back("I8"); //19
  namelist.push_back("I9"); //20
  namelist.push_back("I10"); //21
  namelist.push_back("I11"); //22
  namelist.push_back("I12"); //23
  namelist.push_back("I13"); //24
  namelist.push_back("I14"); //25
  namelist.push_back("I15"); //26
  namelist.push_back("I16"); //27
  namelist.push_back("DATA"); //28
  namelist.push_back("CLK"); //29
  namelist.push_back("SET"); //30
  namelist.push_back("CLEAR"); //31
  namelist.push_back("Q"); //32
  namelist.push_back("QBAR"); //33
}

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;
}
```

```
80
81 namestring names::getnamestring(name id)
82 {
83   if (id > blankname && id < namelist.size()) return namelist[id];
84   else return "";
85 }
```

Listing 2: names.cc

## 1.2 Scanner Class

### 1.2.1 scanner.h

```
1  #ifndef scanner_h
2  #define scanner_h
3  #include <string>
4  #include <iostream>
5  #include <fstream>
6  #include <cstdlib>
7  #include "names.h"
8
9  using namespace std;
10
11 typedef int name;
12 typedef enum {namesym, numsym, devsym, consym, monsym, endsym, classsym, iosym, colon,
       semicol, equals, dot, badsym, eofsym} symbol;
13
14 class scanner
15 {
16   public:
17     symbol s;
18     names* defnames;//Pointer to instance of names class
19
20     scanner (names* names_mod,     //Pointer to names class
21          const char* defname);  //Name of file being read
22     ~scanner();            //Destructor
23     void getsymbol(symbol& s,    //Symbol type read
24               name& id,       //Return symbol name (if it has one)
25               int& num);      //Return symbol value (if it's a number)
26     void getcurrentline();
27
28   private:
29     ifstream inf; //Input file
30     char curch;   //Current input character
31     char prevch;  //Previous input character. Used for finding line end
32     bool eofile;  //True for end of file
33     bool eoline;  //True for line end
34     int linenum;  //Number of lines in definition file
35     int cursymlen;  //Length of current symbol. Used for error printing
36     string line;  //Current line contents. Used for error printing
37
38
39     void getch(); //Gets next input character
40     void getnumber(int& number);  //Reads number from file
41     void getname(name& id); //Reads name from file
42     string getline(); //Reads the line
43     void writelineerror();  //Writes out an error with a caret pointer
44     void skipspaces(); //Skips spaces
45     void skipcomments(); //Skips comments
46 };
47
48 #endif
```

Listing 3: scanner.h

### 1.2.2 scanner.cc

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

```
3
4  using namespace std;
5
6  scanner::scanner(names* names_mod, const char* defname)
7  {
8    defnames = names_mod;
9    inf.open(defname);  //Open file
10   if (!inf)
11   {
12     cout << "Error: cannot open file for reading" << endl;
13   }
14   eofile = (inf.get(curch) == 0); //Get first character
15 }
16
17 scanner::~scanner()
18 {
19   inf.close();  //Close file
20 }
21
22 void scanner::getsymbol(symbol& s, name& id, int& num)
23 {
24   num = 0;
25   s = badsym;
26   cursymlen = 0;
27   skipspaces();
28   skipcomments();
29   if (eofile) s = eofsym;
30   else
31   {
32     if (isdigit(curch))
33     {
34       s = numsym;
35       getnumber(num);
36     }
37     else
38     {
39       if (isalpha(curch))
40       {
41         getname(id);
42         if (id == 0) s = devsym;
43         else if (id == 1) s = consym;
44         else if (id == 2) s = monsym;
45         else if (id == 3) s = endsym;
46         else if (id > 3 && id < 12) s = classsym;
47         else if (id > 11 && id < 34) s = iosym;
48         else s = namesym;
49       }
50       else
51       {
52         switch (curch)
53         {
54           case '=':
55             s = equals;
56             break;
57           case ';':
58             s = semicol;
59             break;
60           case ':':
61             s = colon;
62             break;
63           case '.':
64             s = dot;
65             break;
66           default:
67             s = badsym;
68             break;
69         }
70         cursymlen = 1;
71         getch();
72       }
73     }
74   }
75 }
76
77 void scanner::writelineerror()
78 {
```

```
  string errorptr;
  for (int i = 0; i < (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 (curch == '\n')
  {
    linenum++;
    line.clear();
  }
  if (prevch != '\n')
  {
    line.push_back(prevch);
  }
}

void scanner::getnumber(int& number)
{
  number = 0;
  cursymlen = 0;
  while (isdigit(curch))
  {
    number *= 10;
    number += (int(curch) - int('0'));
    cursymlen++;
    getch();
  }
}

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

void scanner::skipspaces()
{
  while (isspace(curch))
  {
    getch();
    if (eofile) break;
  }
}

void scanner::skipcomments()
{
  if (curch == '/')
  {
    getch();
    if (curch == '*')
    {
      getch();
      while (!(prevch == '*' && curch == '/'))
      {
        getch();
        if (eofile) break;
      }
      getch();
      getch();
```

```
155         getch(); //Get to next useful char
156       }
157     }
158 }
159
160 string scanner::getline()
161 {
162   if (s != semicol)
163   {
164     while (curch != ';' && !eofile)
165     {
166       getch();
167     }
168     line.push_back(curch);
169   }
170   return line;
171 }
```

Listing 4: scanner.cc

## 1.3 Parser Class

### 1.3.1 parser.cc

```
1  void parser::connectionList()
2  {
3    //EBNF: connections = 'CONNECTIONS' [con] {';' con} 'END'
4    smz->getsymbol(cursym, curname, curint);
5    if (cursym == endsym)
6    {
7      return;
8    }
9    else if (cursym == namesym)
10   {
11     newConnection();
12   }
13   else
14   {
15     error();//connection must start with the name of a device
16     cout << "connection must start with the name of a device" << endl;
17   }
18   smz->getsymbol(cursym, curname, curint);
19   while (cursym == semicol)
20   {
21     smz->getsymbol(cursym, curname, curint);
22     if (cursym == namesym)
23     {
24       newConnection();
25     }
26     else if (cursym == endsym)
27     {
28       return;
29     }
30     else
31     {
32       error();//connection must start with the name of a device or end of device list
       must be terminated with END (not semicolon)
33       cout << "connection must start with the name of a device or end of device list must
        be terminated with END (not semicolon)" << endl;
34     }
35     smz->getsymbol(cursym, curname, curint);
36   }
37 }
38
39 void parser::newConnection()
40 {
41   //EBNF: con = devicename '.' input '=' devicename ['.' output]
42   if (smz->defnames->namelength (curname) != 0)
43   {
44     connectionInName = curname;
45     smz->getsymbol(cursym, curname, curint);
46     if (cursym == dot)
```

```
47      {
48        smz->getsymbol(cursym, curname, curint);
49        if (cursym == iosym)
50        {
51          name inputPin = curname;
52          smz->getsymbol(cursym, curname, curint);
53          if (cursym == equals) //SEARCH - you have got to here
54          {
55            smz->getsymbol(cursym, curname, curint);
56            if (smz->defnames->namelength (curname) != baddevice)
57            {
58              connectionOutName = curname;
59              switch (curname)
60              {
61                case 10:
62                  smz->getsymbol(cursym, curname, curint);
63                  if (cursym == dot)
64                  {
65                    smz->getsymbol(cursym, curname, curint);
66                    if (cursym == iosym)
67                    {
68                      netz->makeconnection(connectionInName, inputPin, connectionOutName,
    cursym, correctOperation); //DAT NESTING
69                      return;
70                    }
71                  }
72                  else
73                  {
74                    error();  //Expect a dot after dtype
75                    cout << "Expect a dot after dtype" << endl;
76                  }
77                default:
78                  netz->makeconnection(connectionInName, inputPin, connectionOutName,
    blankname, correctOperation);
79                  return;
80              }
81            }
82            else
83            {
84              error(); //Device does not exist
85              cout << "Device does not exist" << endl;
86            }
87          }
88          else
89          {
90            error();//SEARCH - you have got to here
91            cout << " " << endl;
92          }
93        }
94        else
95        {
96          error();//specify input gate after dot
97          cout << "specify input gate after dot" << endl;
98        }
99      }
100     else
101     {
102       error();//need to seperate connection input with a '.' (or need to specify input)
103       cout << " " << endl;
104     }
105   }
106   else
107   {
108     error(); //Device does not exist
109     cout << "Device does not exist" << endl;
110   }
111 }
112
113 void parser::monitorList()
114 {
115   //EBNF: monitors = 'MONITORS' [mon] {';' mon} 'END'
116   smz->getsymbol(cursym, curname, curint);
117   if (cursym == endsym)
118   {
119     return;
120   }
```

```
121   else if (cursym == namesym)
122   {
123     newMonitor();
124   }
125   else
126   {
127     error();//monitor must start with the name of a device
128     cout << "monitor must start with the name of a device" << endl;
129   }
130   smz->getsymbol(cursym, curname, curint);
131   while (cursym == semicol)
132   {
133     smz->getsymbol(cursym, curname, curint);
134     if (cursym == namesym)
135     {
136       newMonitor();
137     }
138     else if (cursym == endsym)
139     {
140       return;
141     }
142     else
143     {
144       error();//monitor must start with the name of a device or end of device list must
      be terminated with END (not semicolon)
145       cout << "monitor must start with the name of a device or end of device list must be
       terminated with END (not semicolon)" << endl;
146     }
147     smz->getsymbol(cursym, curname, curint);
148   }
149 }
150
151 void parser::newMonitor()
152 {
153 //EBNF: mon = devicename['.'output]
154   if (smz->defnames->namelength (curname) != 0)
155   {
156     monitorName = curname;
157     switch (curname)
158     {
159       case 10:
160         smz->getsymbol(cursym, curname, curint);
161         if (cursym == dot)
162         {
163           smz->getsymbol(cursym, curname, curint);
164           if (cursym == iosym)
165           {
166             mmz->makemonitor(monitorName, cursym, correctOperation);
167             return;
168           }
169         }
170         else
171         {
172           error();  //Expect a dot after dtype
173           cout << "Expect a dot after dtype" << endl;
174
175         }
176       default:
177         mmz->makemonitor(monitorName, blankname, correctOperation);
178         return;
179     }
180   }
181   else
182   {
183     error();
184     cout << "Bad device monitor" << endl;
185   }
186 }
```

Listing 5: parser.cc

# 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 6: xor.gf2

### 2.1.2 Circuit Diagram

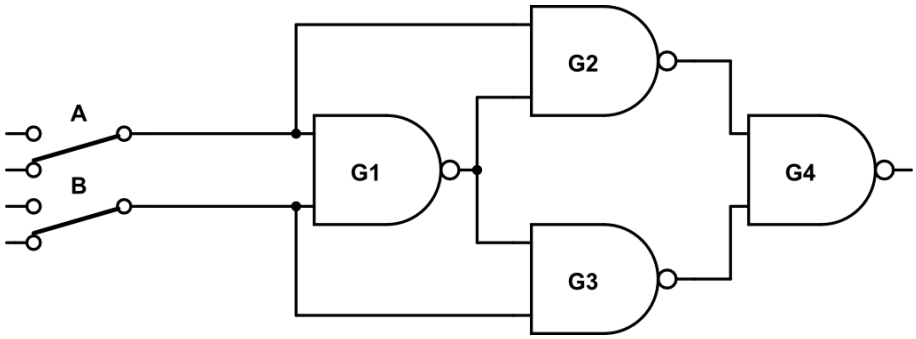


Figure 1: Circuit diagram of an XOR gate implemented using NAND gates

## 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;
OR2.I2 = AND2;

XOR3.I1 = A1;
XOR3.I2 = B1;
AND3.I1 = XOR3;
AND3.I2 = OR1;
AND4.I1 = A1;
AND4.I2 = B1;
XOR4.I1 = XOR3;
XOR4.I2 = OR1;
OR2.I1 = AND3;
OR2.I2 = AND4;

XOR5.I1 = A2;
XOR5.I2 = B2;
AND5.I1 = XOR5;
AND5.I2 = OR2;
AND6.I1 = A2;
AND6.I2 = B2;
XOR6.I1 = XOR5;
XOR6.I2 = OR2;
OR3.I1 = AND5;
OR3.I2 = AND6;

/* MSB Adder */
XOR7.I1 = A3;
XOR7.I2 = B3;
AND7.I1 = XOR7;
AND7.I2 = OR3;
AND8.I1 = A3;
AND8.I2 = B3;
XOR8.I1 = XOR7;
XOR8.I2 = OR3;
OR4.I1 = AND7;
OR4.I2 = AND8;
END

MONITORS
/* Outputs */
XOR2;
XOR4;
```

```
XOR6;
XOR8;
OR4; /* Carry out */
END
```

Listing 7: 4bitadder.gf2

### 2.2.2 Circuit Diagram

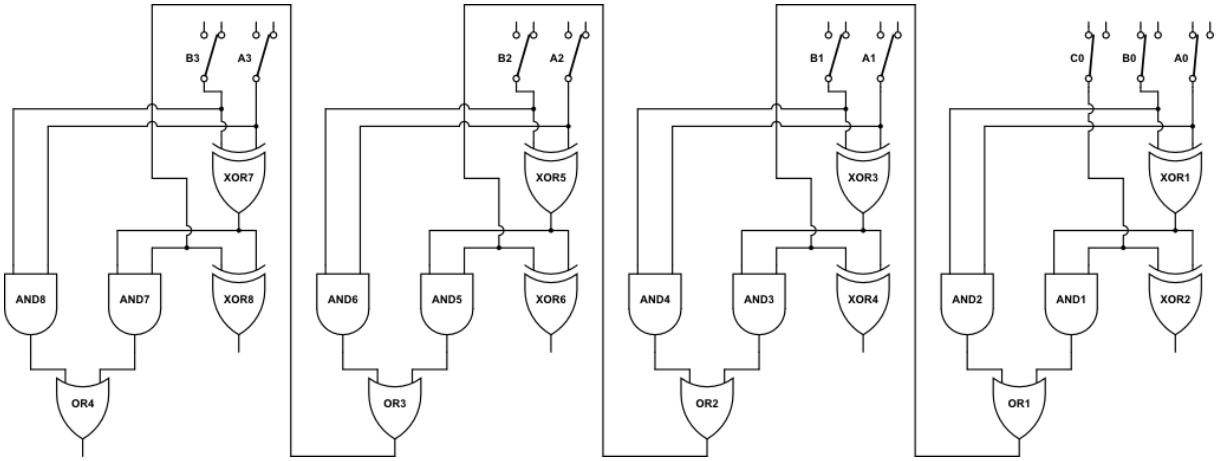


Figure 2: Circuit diagram of a 4-bit adder

## 2.3 Serial In Parallel Out Shift Register

### 2.3.1 Definition File

```
DEVICES
CLOCK CLK1:2;
CLOCK CLK2:1;
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;
END

MONITORS
CLK2;
D1.Q;
D2.Q;
D3.Q;
D4.Q;
END
```

Listing 8: sipo.gf2

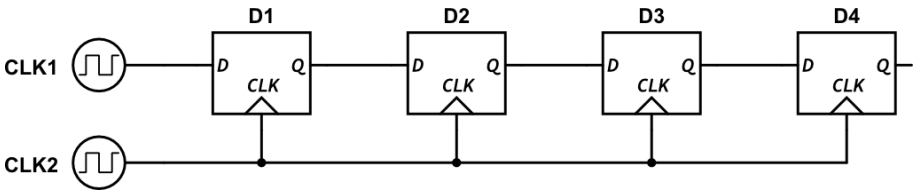


Figure 3: Circuit diagram of a serial in parallel out shift register

### 2.3.2 Circuit Diagram

## 2.4 Gated D Latch

### 2.4.1 Definition File

```
DEVICES
CLOCK CLK1:1;
CLOCK CLK2:2;
NAND G1:1;
AND G2:2;
AND G3:2;
NOR G4:2;
NOR G5:2;
END

CONNECTIONS
G1.I1 = CLK1;
G2.I1 = G1;
G2.I2 = CLK2;
G3.I1 = CLK2;
G3.I2 = CLK1;
G4.I1 = G2;
G4.I2 = G5;
G5.I1 = G4;
G5.I2 = G3;
END

MONITORS
CLK1; /* D */
CLK2; /* E */
G4; /* Q */
G5; /* QBAR */
END
```

Listing 9: sipo.gf2

### 2.4.2 Circuit Diagram

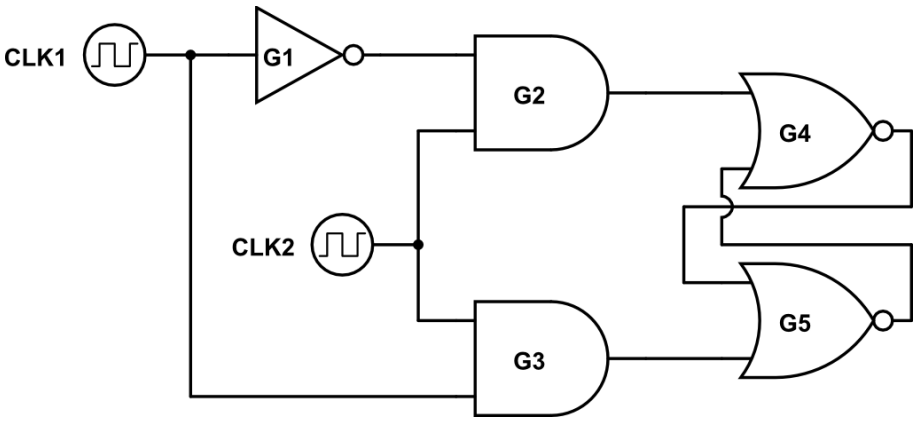


Figure 4: Circuit diagram of a Gated D Latch

# 3 User Guide

test