IIA GF2 Software: 2nd Interim Report

Tim Hillel (th389) - Team 8

1 Code Listings

1.1 Parser Class

1.1.1 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();
    void newDevice(int deviceType);
    void newConnection();
    void newMonitor();
    network* netz; // instantiations of various classes for parser to use.
    devices* dmz;
    monitor* mmz;
    scanner* smz;
    error* erz;
    int curint;
    symbol cursym;
    name curname;
    bool correctOperation;
    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 1: parser.h

1.1.2 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 = 0, connectionDone = 0, monitorDone = 0;
  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
      }
      devicePresent = 0;
      deviceDone = 1;
      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
      }
      connectionPresent = 0;
      connectionDone = 1;
      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
      }
      monitorPresent = 0;
      monitorDone = 1;
      monitorList();
    }
    else
    {
      while (cursym != devsym && cursym != consym && cursym != monsym && cursym != eofsym)
      {
        smz->getsymbol(cursym, curname, curint);
      }
    }
  }
  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'
  if (!devicePresent)
  {
    smz->getsymbol(cursym, curname, curint);
    if (cursym == classsym)
    {
      newDevice(curname);
      devicePresent = 1;
    }
    else if (cursym == endsym)
    {
      erz->newError(3); //must have at least one device
      return;
    }
    else
    {
      erz->newError(4); //need a device type
    }
    smz->getsymbol(cursym, curname, curint);
  }
  while (cursym == semicol)
  {
    smz->getsymbol(cursym, curname, curint);
    if (cursym == classsym)
    {
      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)
    }
    smz->getsymbol(cursym, curname, curint);
  }
  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;
  }
}


void parser::newDevice(int deviceType)
{
  //EBNF: dev = clock|switch|gate|dtype|xor
  smz->getsymbol(cursym, curname, curint);
  if (cursym == namesym)
  {
    name devName = curname;
    if (deviceType == 10)
    {
      dmz->makedevice(dtype, devName, 0, correctOperation); //create DTYPE with name devName
      return;
    }
    if (deviceType == 11)
    {
      dmz->makedevice(xorgate, devName, 2, correctOperation); //create XOR with name devName
      return;
    }
    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
            }
            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
            }
            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
            }
            break;
          default:
            cout << "Please do not deduct marks if this message is displayed" << endl;
        }
        return;
      }
      else
      {
        erz->newError(9);//clock needs clock cycle number
      }
    }
    else
    {
      erz->newError(10);//need colon after name for CLOCK/SWITCH/GATE type
    }
  }
  else
  {
    erz->newError(11);//name must begin with name starting with letter and only containing letter
    number and _
  }
}

void parser::connectionList()
{
  //EBNF: connections = 'CONNECTIONS' {con ';'} 'END'
  if (!connectionPresent)
  {
    smz->getsymbol(cursym, curname, curint);
    if (cursym == endsym)
    {
      erz->newWarning(0);//No Connections
      return;
    }
    else if (cursym == namesym)
    {
      newConnection();
      connectionPresent = 1;
    }
    else
    {
      erz->newError(12);//connection must start with the name of a device
    }
    smz->getsymbol(cursym, curname, curint);
  }
  while (cursym == semicol)
  {
    smz->getsymbol(cursym, curname, curint);
    if (cursym == namesym)
    {
      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)
    }
    smz->getsymbol(cursym, curname, curint);
  }
  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;
  }
}

void parser::newConnection()
{
  //EBNF: con = devicename'.'input '=' devicename['.'output]
  if (smz->defnames->namelength(curname) != 0)
  {
    connectionInName = curname;
    smz->getsymbol(cursym, curname, curint);
    if (cursym == dot)
    {
```

```cpp
      smz->getsymbol(cursym, curname, curint);
      if (cursym == iosym)
      {
        name inputPin = curname;
        smz->getsymbol(cursym, curname, curint);
        if (cursym == equals) //SEARCH - you have got to here
        {
          smz->getsymbol(cursym, curname, curint);
          if (smz->defnames->namelength(curname) != 0)
          {
            connectionOutName = curname;
            devlink devtype = netz->finddevice(curname);
            switch (devtype ? devtype->kind : baddevice)
            {
              case 7:
                smz->getsymbol(cursym, curname, curint);
                if (cursym == dot)
                {
                  smz->getsymbol(cursym, curname, curint);
                  if (cursym == iosym)
                  {
                    netz->makeconnection(connectionInName, inputPin, connectionOutName, curname, correctOperation);
                    return;
                  }
                }
                else
                {
                  erz->newError(14);  //Expect a dot after dtype
                }
              default:
                netz->makeconnection(connectionInName, inputPin, connectionOutName, blankname, correctOperation);
                return;
            }
          }
          else
          {
            erz->newError(15); //Device does not exist
          }
        }
        else
        {
          erz->newError(16);//Must specify output to connect to input with equals sign
        }
      }
      else
      {
        erz->newError(17);//specify valid input gate after dot
      }
    }
    else
    {
      erz->newError(18);//need to seperate connection input with a '.' (or need to specify input)
    }
  }
  else
  {
    erz->newError(19); //Device does not exist
  }
}

void parser::monitorList()
{
  //EBNF: monitors = 'MONITORS' {mon ';'} 'END'
  if (!monitorPresent)
  {
    smz->getsymbol(cursym, curname, curint);
    if (cursym == endsym)
    {
      erz->newWarning(1);//No Monitors
      return;
    }
    else if (cursym == namesym)
    {
      newMonitor();
      monitorPresent = 1;
    }
    else
    {
      erz->newError(20);//monitor must start with the name of a device
    }
    smz->getsymbol(cursym, curname, curint);
  }
  while (cursym == semicol)
  {
    smz->getsymbol(cursym, curname, curint);
    if (cursym == namesym)
    {
      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)
    }
    smz->getsymbol(cursym, curname, curint);
  }
  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;
  }
}

void parser::newMonitor()
{
  //EBNF: mon = devicename['.'output]
  if (smz->defnames->namelength(curname) != 0)
  {
    monitorName = curname;
    devlink devtype = netz->finddevice(curname);
    switch (devtype ? devtype->kind : baddevice)
    {
      case 7:
        smz->getsymbol(cursym, curname, curint);
        if (cursym == dot)
        {
          smz->getsymbol(cursym, curname, curint);
          if (cursym == iosym)
          {
            mmz->makemonitor(monitorName, curname, correctOperation);
            return;
          }
        }
        else
        {
          erz->newError(22);  //Expect a dot after dtype
        }
      default:
        mmz->makemonitor(monitorName, blankname, correctOperation);
        return;
    }
  }
  else
  {
    erz->newError(23);
  }
}
```

```
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 2: parser.cc

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

1.2 Error Class

1.2.1 error.h

```
#ifndef error_h
#define error_h

#include "scanner.h"

typedef string errorstring;

class error
{
  private:
    int errorCount;
    int warningCount;
    vector<errorstring> errorlist;
    vector<errorstring> warninglist;
    scanner* smz;
  public:
    error(scanner* scanner_mod);
    void newError(int errorCode);
    void newWarning(int warningCode);
    bool anyErrors();//outputs total number of errors and warnings and returns 1 if any errors are
    present
};

#endif /* error_h */
```

Listing 3: error.h

1.2.2 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 iput gate seperated 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: Devices block must be initialised with 'DEVICES'"); //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 'MONITOR' block, it may not have been initialised properly");//31
  errorlist.push_back("Error 0x0020: Block must be terminated with 'END'");//32
  errorCount = 0;
  warningCount = 0;
  warninglist.push_back("Warning 0x0000: You have not specfied any conenctions. Please check this is what is required");//0
  warninglist.push_back("Warning 0x0001: You have not specfied 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 ++;
}

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 4: error.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 5: xor.gf2

2.1.2 Circuit Diagram

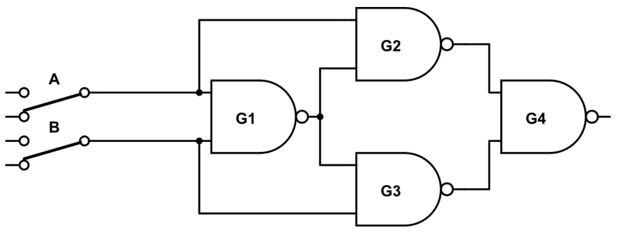


Figure 1: Circuit diagram of an XOR gate implemented using NAND gates

2.2 4-bit Adder

2.2.1 Definition File

```
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 6: 4bitadder.gf2

2.2.2 Circuit Diagram

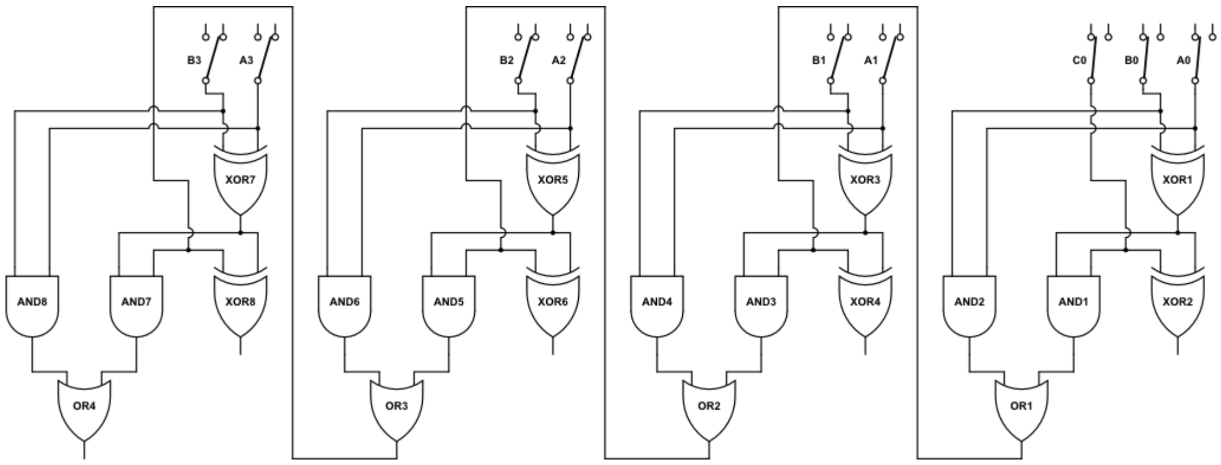


Figure 2: Circuit diagram of a 4-bit adder

2.3 Serial In Parallel Out Shift Register

2.3.1 Definition File

```
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 7: sipo.gf2

2.3.2 Circuit Diagram

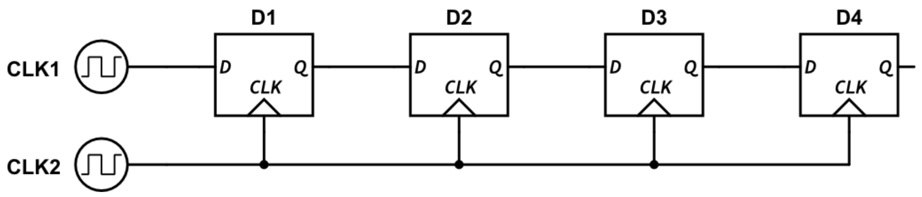


Figure 3: Circuit diagram of a serial in parallel out shift register

NB The software used to draw the circuit diagram does not support the same style of D flip-flop used in the definition file, and Fig. 3 was the closest achievable.

2.4 Gated D Latch

2.4.1 Definition File

```
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 8: sipo.gf2

2.4.2 Circuit Diagram

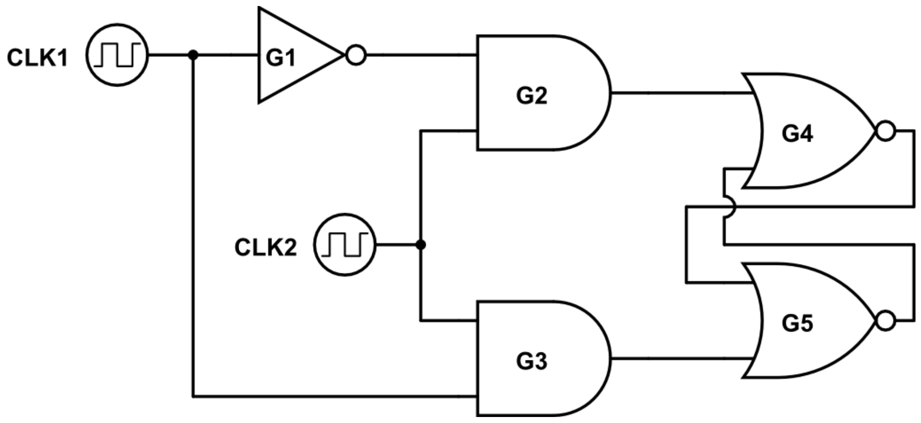


Figure 4: Circuit diagram of a Gated D Latch

NB The software used to draw the circuit diagram does not support the NAND gates with one input. Therefore the NAND gate G1 was substituted for a NOT gate as can be seen in Fig. 4.

3 User Guide

To start the logic simulator, open a terminal window and browse to the `src` folder. Start the application by typing `./logsim` followed by the return key. You will then be presented with the default view.

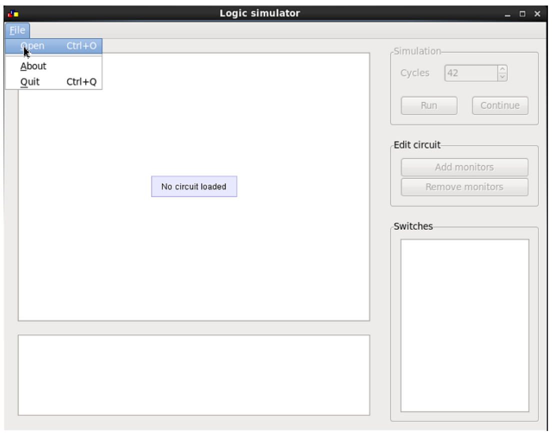
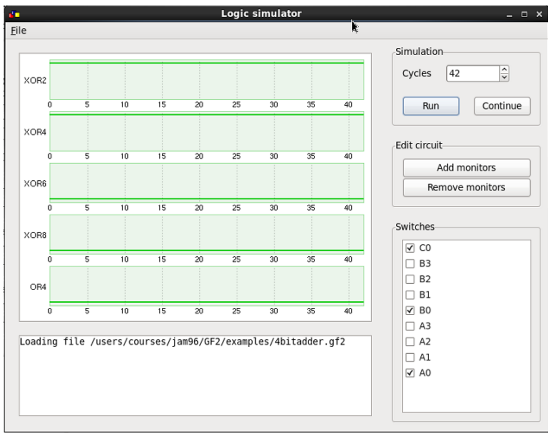


Figure 5: The default view upon opening the Logic Simulator

To open a definition file, click the `File` menu followed by the `Open` option. You will be presented with a file selection dialogue. The file selection dialogue will only show definition files (Files with the `.gf2` file extension). Upon selecting a file, any errors in the definition file will be written to the message window, otherwise the Logic Simulator is ready to use.

In order to run a simulation you must first enter a number of cycles you wish the simulation to run for (default is 42) then press the run button. The monitored signals will be displayed in the left display panel. You may choose to continue the simulation by pressing the continue button.



```
Loading file /users/courses/jam96/GF2/examples/4bitadder.gf2
```

Figure 6: The view upon running a simulation

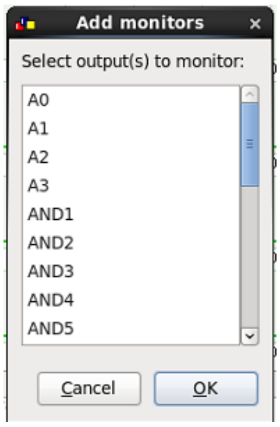


Figure 7: Add monitors dialogue

You can also edit the monitors from within the logic simulator. To add monitors, click the `Add monitors` button and select the monitor, or monitors, you wish to add followed by the `OK` button. To remove monitors, press the `Remove monitors` button and select the monitor, or monitors, you wish to remove followed by the `OK` button

In addition, if your circuit contains any switches, you can change the state of the switch by changing the state of the check box beside its name.