LogSim

Final Project Report

December 20, 2005

Nithya S Ganesan (nsg2104), Mukul Khajanchi (mk2603), David Lau (dsl2012), Xunzhou Zhang (xz2025)
PLT Fall 2005

Chapter 1- White paper

1.1 Background

Digital electronics is an indispensable part of our lives today. Televisions, computers, and cars all depend on digital electronics. In the field of digital design, engineers often encounter the issue of design and testing. However, building each circuit physically and testing it can be very costly. Hence, they need tools that enable them to describe their design abstractly. This gives rise to the need for Hardware Description Languages (HDL), that enable the programmers to describe a digital circuit efficiently, verify that it's correct, and perform virtual simulations on the design.

Currently, there are two major HDL's in the market, VHDL and Verilog. VHDL, which was created by the United States Department of Defense in 1981, was the first HDL created, and is still widely used today. Verilog, created 2 years later by Gateway Design Automation Inc., as a direct competition to VHDL, is another popular HDL. Both VHDL and Verilog are hugely complex languages that require a very steep learning curve. Though they are powerful, programmers often fail to exploit all their features due to their complex syntaxes. Our goal is to build a simple and efficient language that can be used to simulate logic circuits, and yet maintains most of VHDL and Verilog's functionality.

1.2 Features

1.2.1 Portability

One advantage of LogSim is portability. We will use ANTLR as the syntax recognizer, which runs on a JVM. Since we are using ANTLR, LogSim compiler will generate Java code that can be compiled by Java compiler in JDK and run on the JVM.

1.2.2 Ease of Use

LogSim will have a simple and intuitive syntax that is quick to learn and easy to decipher. Programmers will be able take advantage of all the features of LogSim without being overwhelmed by syntax issues. Electrical engineers who do not have too much programming background can easily use it for simulating their designs.

1.2.3 Efficiency

Since this is a small, domain-specific language, it will have a small set of keywords, identifiers and operators. This ensures that implementation can be made as efficient as possible, and program compilation will be quick and with minimal system resources utilization.

1.3 Language Description

A LogSim program consists of operators and components. Operators include basic gatessuch as AND, OR, NOR, XOR, etc. Components include MUXes, Encoders, Latches, Flipflops, etc, that the user can make use of in the code. In addition to the built-in components, users can define their own components using LogSim syntax. Operators and components together make up a System. A System is the final circuit that the user specifies and all simulations are run on the System.

1.3.1. Sample Program

Consider the implementation of a one-bit Full Adder where the Sum and Carry are calculated as:

Sum = (A xor B) xor Cin
CarryOut = (A and B) or (Cin and (A xor B))

This can be implemented using LogSim as a component. Using this component, a two bit Full Adder can be built easily and the System can be simulated.

Sample code using LogSim:

```
//signal keyword defines a one-bit variable
Component FullAdder(In: A,B,Cin; Out: Sum, Cout)
{
      Sum = (A # B) # Cin; // # means xor
      Cout = (A * B) + (Cin * (A # B)); // *, + mean and, or
}
Component TwoBitAdder(In: A0,A1,B0,B1; Out: S0,S1,Cout)
{
      Signal C_temp;
      fullAdder(A0, B0; S0, C_temp);
      fullAdder(A1, B1 C_temp; S1, Cout);
}
//System keyword denotes the main program, where the simulation occurs
System()
{
      A0=^0;
      A1=^1;
      B0=^1;
      B1=^0;
      //Calls the TwoBitAdder function with the appropriate input and
        output variables.
      TwoBitAdder(A0,A1,B0,B1; S0,S1,Cout);
}
```

1.4 Summary

LogSim enables easy programming and simulation of logic circuits, and is a lightweight language that can be readily deployed for use.

Chapter 2 - Tutorial

All LogSim programs print the inputs and outputs for the System component. The System Component is the final, top-layer circuit

2.1 Variable Declaration

To declare a variable, simply write the variable name and assign it a value. The value can be any number of bits;

Example:

```
MyVar =  ^10101;   //Assigning “10101” combinatorially to MyVar
MyVar := ^10101;   //Assigning “10101” sequentially to MyVar
```

It is also possible to extract certain bits from a variable, using the “vector slicing” feature of LogSim.

Example:

NewVar = MyVar [0..3]; //stores the first 4 bits into NewVar, NewVar would be “1010”

LogSim also allows you to truncate or expand a variable to store any number of bits you desire, all you have to do is put a $ followed by the number of bits you want to allocate after your variable.

Example:

```
Expanded_Var$8 = ^1111;      //Allocates 8 bits to Expanded_Var, this
                              will store “00001111” into Expanded_Var

Truncated_Var$4 = ^00001111; //Allocates 4 bits to Truncated_Var, this
                              will store “1111” to Truncated_Var
```

2.2 Component Declaration

A component simply takes in a set of input variables and performs some operations on them and stores the output to an output variable.

Example:

```
Component MyComponent (In: input1, input2, input2; Out: output1)
{    //  Your code for MyComponent goes here.  }
```

MyComponent is a user-defined name of the component, with “input1”, “input2”, “input3” specified as inputs, and “output1” as output

2.3 The System Component

Each LogSim program must have a component named System with no input or output parameters. Inside the System component, previously-declared components are instantiated.

2.4 Simple program

Let's write a program that returns a sum-of-products from four input variables, and let's call it `tutorial.ls`

```
//A,B,C,D are variables to the "Expr" Component, and E is the output

Component Expr(In: A,B,C,D; Out:E){
     E=A*B+C*D;    //This is equivalent to E = (A and B) OR (C and D)
}

//System keyword denotes the main program, where the simulation occurs
System ()
{
  A=^1111;
  B=^0000;
  C=^1111;
  D=^1001;

  Expr(A,B,C,D;E)  // This sends the variables A,B,C,D
                      into the Expr component, and stores the output of
                      the component to E.
}
```

To run this program, simply type at the command prompt: `./lsim tutorial.ls 1` (Note: the "1" means that the program should display variables for one clock cycle)

This program will simply output:

```
Tick 0:
D = 1 0 0 1     A = 1 1 1 1     C = 1 1 1 1     B = 0 0 0 0     E = 1 0 0 1
```

Note: The output prints the final results for all variables that are included within the System() component, in non-deterministic order.

Chapter 3 - Language Reference Manual

3.1 Lexical Conventions

3.1.1 Comments

3.1.1.1 Multi-line comments

The /* characters introduce a comment; the */ characters terminate a comment. Once a /* is seen, all other characters are ignored until the ending */ is encountered.

3.1.1.2 Single-line comments

All the text after // until the end of line are ignored

3.1.2 Identifiers

An *identifier* is an unlimited-length sequence of letters, digits and underscores, the first of which must be a letter. An identifier cannot be the same as a keyword.
Examples:

- `Abcd_123;` // `Abcd_123` is a valid identifier `12_abc` // 12_abc is an invalid because identifiers cannot start with a number
- `component;` // component is an invalid identifier because it is a keyword

3.1.3 Keywords

The following character sequences are reserved for use as *keywords* and cannot be used as identifiers:

```
in out import
```

3.1.4 Constants

3.1.4.1 Constants

3.1.4.1.1 Decimal Constants

Default is in decimal format. For example, 12 means 1100

3.1.4.1.2 Binary Constants

Of the format ^1100

3.1.5 Operators

An operator specifies an operation to be performed. The operators ( ) and [ ] must occur in pairs, possibly separated by expressions.
Operator can be one of the following: ( ) [ ] + * # != := $

3.1.6 Punctuators

A punctuator is a symbol that has semantic significance but does not specify an operation to be performed. The punctuators ( ), [ ], and { } must occur in pairs, possibly separated by expressions, declarations, or statements. The following punctuators are used in LogSim: ; , () {} [ ]

3.2 Expressions and Operators

3.2.1 Precedence and Associativity

( ) [] $	Component Instantiation, Vector indexing, Truncation	Post-fix	L-R
…	Vector slicing	Unary	L-R
!	NOT	Pre-fix Unary	L-R
*	And	Binary	L-R
#, +	Xor, OR	Binary	L-R
:=, =	Sequential Assignment, Combinational Assignment	Binary	R-L

3.3 Types

3.3.1 Vectors

Vectors are the only data type in LogSim. A single bit signal is simply a one member vector. Vector members can be accessed using an index within brackets. For example A[2] refers to the second element in the vector A. To declare a vector, simply write the vector name followed by an equal sign and the value of the vector. For example, A=101 defines a vector of the binary value '101'. By default, if a vector is referred to without an index, it points to the first member of the vector.

3.3.1.1 Vector Slicing

Any continuous subset of a vector can be accessed using the following syntax: `A[n...n+k]` returns all the bits from the nth element to the (n+k)th element in A.

3.4 Components

Components are the basic building blocks in LogSim. A Component with the name System is the one on which simluations will be run.

3.4.1 Declaring a Component

`Component` *`Component_Name`* `(In:` *`I0,I1,I2...`*`; Out:` *`O1,O1,O2...`*`)`

I0, I1, I2,.... are inputs to the component, and *O1,O2,O3...* are outputs to the component.

3.5 Library

A library is a logsim source file that does not contain a System component. It is used in conjunction with the `import` keyword to specify that all components inside the library file can be directly accessed in the main program. The library file name should end with an "lib" extension.

Example:

```
import MyComponent.lib
```

This statement specifies that all components defined in the MyComponent.lib file are accessible in the main program.

3.6 Statements

3.6.1 Sequential Assignment

To assign values sequentially, use the ":=" operator.

Example:

```
A$8:=B;
B$8:=C;
```

will assign B to A in clock cycle 1, and C to B in clock cycle 2, so that A in does not immediately take the value of C, as it would in combinational assignments. The “$8” after the variable name allocates 8 bits for the variable.

Note: If the result has more than 8 bits, it’ll be right-truncated. If the result has fewer than 8 bits, the most significant bits will be filled with 0’s.

3.6.2 Combinational Assignment

To assign values combinationally, use the "=" operator. For example: `A=11` will assign the binary value '11' to the vector A. Note: all combinationally assignments in LogSim are concurrent.

Example:

```
A$8=B;
B$8=C;
```

will assign the value of C to A, as all assignments are evaluated concurrently.

3.6.3 Component Instantiation

A component is instantiated from within another component. Upon each instantiation, a new copy of the component is created.

3.7 Output

Upon running the program, the program will print to the screen the values of all inputs and outputs to the System Component for one clock cycle. If the user wishes to see values for more than one clock cycle, he may enter the name of the program followed by two integers that specify the starting and ending clock cycles that the user wants to see.

Example:

```
./lsim MyProgram.ls 1 100 //Prints inputs and outputs in the System
                            Component from the 1st clock cycle to the 100th
                            clock cycle
```

3.8 Namespaces

LogSim has two namespaces, one for Vectors and one for Components. A Vector can have the same name as a Component.

3.9 Scopes

All components are accessible from everywhere within the same file. All vectors are accessible only from within the component that they are defined in. Within a component, only vectors declared inside and passed through are accessible. All vectors are passed by reference.

Chapter 4 - Project planning

4.1 Project planning

Since this is a long-term project spanning an entire semester, our group decided that it was necessary not to fall behind early, so we set up a meeting every Monday at 5 p.m. During these meetings we reported to each other on our progress, and tried our best to answer each other's questions.

4.2 Team responsibilities

We realized that two of our team members were more advanced in their knowledge of computer science, and thus we divided our group into two teams of two. The division of labor is as follows:

Team 1: Joe and David	• Parser • Testing • Debugging • Documentation- proposal, LRM, final report
Team 2: Nithya and Mukul	• Lexer • Abstract syntax tree • Topologicalsorting • Depth first search • Implementation of functionality

These were our initial responsibilities; in other words, each team is responsible for their respective tasks, but we helped each other and contributed to all tasks together.

4.3 Project history

9/5/05	Team formed
9/15/05	Devised the basic syntax and semantics of LogSim
9/23/05	White paper submitted
10/15/05	Sample LogSim program written
10/28/05	LRM submitted
11/5/05	Lexer completed
11/10/05	Parser completed
12/05/05	Tree walker completed
12/11/05	Completed the topological sort
12/15/05	Documentation and Regression testing
12/21/05	Project submitted

Chapter 5 - Architecture and design

5.1 Overview

The LogSim interpreter has several major blocks that are common, i.e. lexer, parser, tree walker. In addition to this, to implement the concurrent execution of combinational nodes, there is a component which topologically sorts the operators, after the walker build the tree. The output is the state of all the signals in the System component, in each cycle(The number of cycles can be specified by the user). The input is a LogSim file, which has the .ls extension by convention. Fig. 5.1 shows an overview of our translator.

The entry point of the interpreter is the class Main. However, we have provided a small script based wrapper around this class, called lsim, which takes care of concatenation of imported files and calls the Main class providing the appropriate input.

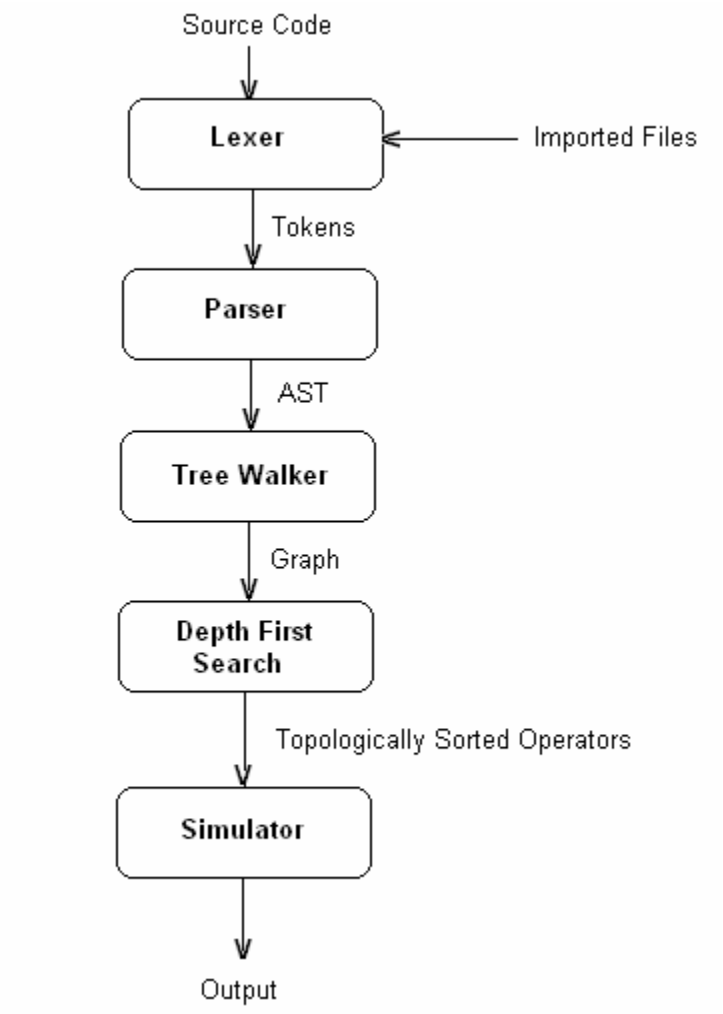

Fig. 5.1 Architecture

5.2 Design

Every variable in `LogSim`, has an object of `LogSimVar` associated with it. Each `LogSimVa`r has two attributes, its name, and the value, which is a `LogSimAtom`. The `LogSimAtom` is the class which provides the implementation of all the operators. However, when the tree is being walked, the walker creates an object of the type `Operato`r. The `Operator` class derives from a class `Nod`e, which models a node in a graph and hence, maintains its adjacency list and is capable of doing a Depth First Search. Every operator has to have the methods `setOutput`, and `evaluate`. Since there can be either binary or unary operators, we have created a base class for each of them, namely `BinOperator` and `UnOperator.` Fig 5.2 shows the hierarchy of operators in LogSim.

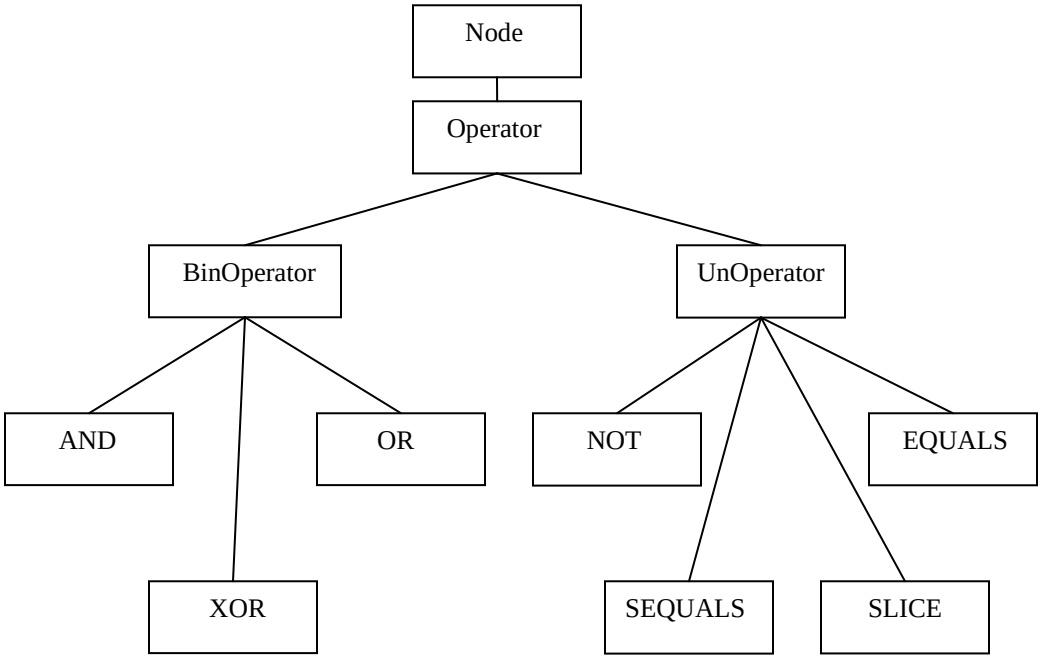

Fig 5.2 Hierarchy of Operators

Chapter 6 - Testing

6.1 Preliminary testing

6.1.1 Lexer & Parser

Our preliminary test was to make sure that the lexer was scanning the correct tokens and the parser was creating the correct abstract syntax tree. To do this, we wrote a program that included all conceivable features of our language and ran it to make sure it produced the correct AST. At this point our language was not fully implemented yet, so we could only confirm that the lexer and parser were correct. The following is this test program:

```
Component test(In: i; Out: o){
	o = i;
}

Component FullAdder(In: A,B,Cin; Out: Sum, Cout)
{
	Sum = (A # B) # Cin; // # means xor
	Cout = (A * B) + (Cin * (A # B)); // *, + mean and, or
}
Component TwoBitAdder(In: A0,A1,B0,B1; Out: S0,S1,Cout)
{
	fullAdder(A0, B0, ^0, S0, Ctemp);
	fullAdder(A1, B1, Ctemp, S1, Cout);
}

//System keyword denotes the main program, where the simulation occurs
System ()
{
	A0=0;
	A1=1;
	B0=1;
	B1=0;
	//Display function displays the results of the component, along
	//with inputs
	TwoBitAdder( A0,A1,B0,B1;S0,S1,Cout);
}
```

6.1.2 Functionality Testing

In this final phase of testing, we wrote several test programs to test every possible functionality of our program. We separately tested individual modules of our language (eg, combinational assignments, sequential assignments, vector slicing, component declaration and instantiation)

6.2 Regression testing

After the preliminary testing, the next step was to automate the testing for the language, so that everytime a change was made, it would be very easy to make sure that the change did not introduce any new errors elsewhere in the language.

To do this, we wrote a script that runs the testing code and outputs the results back to individual text files. Then after changes were made to our langugae, we run that script, and use the “diff” command to check if the outputs are correct.

6.3 Test Modules

test-p-operators.ls

```
Component comp1(In:in1,in2,in3;Out:out1,out2){
  MyVariable=1*1+1#1+!1;
}

System(){
}
```

test-p-cycle.ls

```
System(){
        A = C;
        C = A + B;
        B = 1;
}
```

test-p-index.ls

```
System(){
        A=^1010;
        B=^1010;
        A=A[1];
        B=B[2];
}
```

test-p-size.ls

```
System(){
        B=^1111000011110000;
        A$8 = B;
}
```

test-p-vector.ls

```
System(){
        A = $8;
}
```

test-p-paren.ls

```
System(){
        A = (^1001 # ^1100) * ^1100;
```

```
        B = ^1100 + (^1100 * ^0110);
}
```

test-p-seq.ls

```
System(){
        A=111;
        B=101;
        D=100;

        A := B;
        B := C;
        C := D;
}
```

test-p-comments.ls

```
Component comp1(In: in1,in2,in3; Out: out1,out2){
//comments 1
/* single line comment */
/* multiple line comments
   multiple line comments2
*/

/* //type one comment within type 2 comment */
}

System(){

}
```

test-p-multiplexer.ls

```
Component Multiplexer(In: X0, X1, C; Out: Y){
        Y = (C * X0) + (!C * X1);
}

System(){
        i1 = 1;
        i1 := !i1;

        i2 = 0;
        i2 := !i2;

        c := !c;
        Multiplexer(i1, i2, c; out);
}
```

6.4 Test programs

6.4.1 ALU

```
Component ALU (In: A, B, control; Out: C, C0) {

        /* Control codes for ALU
```

```
            00-add
            01-negate
             10-shift left
             11-shift right
        */

        // CALCULATE NEGATE

        Negate = !A;

        // CALCULATE SUM

        // MSB

        Sum[0] = A[0] # B[0];
        Carry[1] = (A[0] * B[0]) + (^0 * (A[0] # B[0]));

        Sum[1] = A[1] # B[1] # Carry[1];
        Carry[2] = (A[1] * B[1]) + (Carry[1] * (A[1] # B[1]));

        Sum[2] = A[2] # B[2] # Carry[2];
        Carry[3] = (A[2] * B[2]) + (Carry[2] * (A[2] # B[2]));

        // LSB
        Sum[3] = A[3] # B[3] # Carry[3];
        C0 = (A[3]*B[3])+(Carry[3]*(A[3]+B[3]));

        // CALCULATE SHIFT LEFT
        SLL=^0000;
        C0 = A[3];
        SLL[0] = 0;
        SLL[1] = A[0];
        SLL[2] = A[1];
        SLL[3] = A[2];

        // CALCULATE SHIFT RIGHT
        SRL=^0000;
        SRL[3] = 0;
        SRL[2] = A[3];
        SRL[1] = A[2];
        SRL[0] = A[1];

        // CALCULATE WHICH ONE SHOULD BE RETURNED BASED ON
        CONTROL BIT
        C = (SRL * control[0] * control[1]) +
            (SLL * !control[0] * control[1]) +
            (Sum * !control[0] * !control[1]) +
            (Negate * control[0] * !control[1]);
        }

System()
{
  A=^1010;
  B=^1010;
  control=^00; //control codes for add
  ALU(A,B,control;Output,Carry);
}
```

6.4.2 4-Bit Cyclic Counter

```
Component counter(In:not_used; Out:A,B,C,D){
  D:=!D;
  C:=!C * D + C*!D ;
  B:=!B * C * D + B * !(C*D);
  A:=!A * B *C *D + A * !(B*C*D);

}

System(){
  counter(0;A,B,C,D);
}
```

6.4.3 Multiplexer

```
Component Multiplexer(In: X0, X1, C; Out: Y){
        Y = (C * X0) + (!C * X1);
}

System(){
        i1 = 1;
        i1 := !i1;

        i2 = 0;
        i2 := !i2;

        c := !c;
        Multiplexer(i1, i2, c; out);
}
```

Chapter 7 - Lessons learned

7.1 Joe

Although I had worked in groups before, this is my first time participating in a semester long project. I was very excited and looking forward to actually producing a working language. I was put in charge of testing and documentation. This was a very challenging task, as testing is very crucial in the development of a new language. We found many bugs, and this was much different than debugging a normal small program. The scale of everything was much larger than anything I had worked with previously. But I had fun, and got to know my teammates better. This was a great way to bond and work together as a team and as friends.

7.2 Dave

Overall, I felt that this project was a very enriching and educational process. It was the first time I had worked with a group on such a large scale project. I learned many lessons, such as starting early and communicating with teammates is very crucial. I also realized that it is most efficient in a large group project if everyone does what they are best at doing. Since I had a strong background in logic simulation languages such as VHDL, I became one of the testers, using the language to build useful examples and making sure everything was working properly. I am very proud of what we accomplished, although I felt like we were a bit rushed at the end. The important thing, however, was to break large tasks into smaller, manageable tasks. This way you won't feel overwhelmed by the magnitude of what needs to be accomplished.

7.3 Mukul

It was an exciting as well as challenging opportunity to create a language of our own, decide the semantics and all the things user could do in it. During this project, I learnt that it is very important to plan and co-ordinate things well. It is very important to divide the work well.

7.4 Nithya

It was great to see the language working. It was fun designing our own language. But there were a lot more stuff we could have added to make it fully functional. We should have met more frequently (which turned out to be nearly impossible considering each persons timings). I have worked in groups projects before but with people with same background and timings, so this was quiet a challenge.

Appendix

A.1 Lexer

```
class LogSimLexer extends Lexer;

options {
     k=2; // lookahead
     testLiterals = false;
     charVocabulary='\u0000'..'\u007F'; // allow ascii
     exportVocab=LogSim;
}

LPAREN    : '(';
RPAREN    : ')';
SCURL          : '{';
ECURL          : '}';
SARR      : '[';
EARR      : ']';
EQUALS    : '=';
SEQUAL    : ":=";
FSLH      : '/';

OR        : '+';
AND       : '*';
NOT       : '!';
XOR       : '#';

BITINDICATOR : '$';
SPLIC          : "...";
SEMI      : ';';
COMMA          : ',';
COLON          : ':';

protected ALPHA     : 'a'..'z'|'A'..'Z';
protected DIGIT     : '0'..'9';


IDENTIFIER
options {
     testLiterals = true;
}

          : (ALPHA)(ALPHA | DIGIT | '_')* ;
protected
```

```
INT       : (DIGIT)+ ;

protected
BIN       : '^' ('0'|'1')+;
VECTOR         : (INT)|(BIN);

protected
NEWLINE   : ('\n' | ('\r' '\n') => '\r' '\n' | '\r')
            { $setType(Token.SKIP); newline(); };

// rule for single and multiline comments (borrowed from Mx)
COMMENT   : ( "/*" (
                options {greedy=false;} :
                (NEWLINE)
                | ~( '\n' | '\r' )
                )* "*/"
          | "//" (~( '\n' | '\r' ))* (NEWLINE)
          ) { $setType(Token.SKIP); } ;

WS        : ( ' ' | '\t' | NEWLINE)
            {$setType(Token.SKIP);}
;
```

A.2 Parser

```
class LogSimParser extends Parser;


/* EXPR, MIDEXRP, LOWEXPR, ATOM form the grammar for any
expression, like (!A+!B). It assumes right-associative
operators, and it follows the precedence rules of our LRM
*/
options {buildAST=true;}

tokens {
     COMPONENTS;
     COMPONENT;
     IMPORTS;
     FILE;
        STATEMENTS;
     INPUTARGS;
     OUTPUTARGS;
     }
file:
     (importBlock)?
     (componentBlock)
     (systemBlock)?
```

```
     (EOF!)
     { #file = #([FILE, "File"], file);  };

importBlock:
     (importStatement SEMI!)+
     { #importBlock = #([IMPORTS, "Imports"],
importBlock);  };

importStatement:
        "Import"^ IDENTIFIER
     ;

componentBlock:
     (component)*
     { #componentBlock = #([COMPONENTS, "Components"],
componentBlock);  };

component:
     "Component"^ IDENTIFIER LPAREN! in out RPAREN!
statements;

in:
     "In"^ COLON! IDENTIFIER ( COMMA! IDENTIFIER)* SEMI!;

out:
     "Out"^ COLON! IDENTIFIER ( COMMA! IDENTIFIER)*;

statements:
        SCURL! (statement SEMI!)* ECURL!
        {#statements = #([STATEMENTS, "Statements"],
statements); };

statement:
        ( instantiationStatement ) =>instantiationStatement
        | assignmentStatement
     ;
// instantiationStatement:
//        IDENTIFIER^ (LPAREN! "In" COLON! expr ( COMMA!
expr)* SEMI! "Out" COLON! IDENTIFIER (COMMA! IDENTIFIER)*
RPAREN!);

instantiationStatement:
        IDENTIFIER^ (LPAREN! inputArgs SEMI! outputArgs
RPAREN!);

inputArgs:
     expr ( COMMA! expr)*
```

```
     {#inputArgs = #([INPUTARGS, "InputArgs"],
inputArgs); };

outputArgs:
      IDENTIFIER (COMMA! IDENTIFIER)*
     {#outputArgs = #([OUTPUTARGS, "OutputArgs"],
outputArgs); };

// assignment: IDENTIFIER (BIT! DIGIT)? (EQUALS^ | SEQUAL^)
(IDENTIFIER | BIN | expr) SEMI!; //grammar for variable
assignment. For example: A$8=11111111; A:=B; A=B*C, etc etc.
assignmentStatement:
        IDENTIFIER (BITINDICATOR! INT)? (EQUALS^ | SEQUAL^)
expr;

expr:
        midexpr ((XOR^|OR^) midexpr)*;

midexpr:
        lowexpr ((AND^) lowexpr)*;

lowexpr:
        (NOT^ atom) | atom;

atom:
        IDENTIFIER | VECTOR | SLICE | (LPAREN! expr
RPAREN!);

slice:
        IDENTIFIER SARR! DIGIT SPLIC! DIGIT EARR!;
//grammar for vector slicing: A[1..2], this is treated as
an atom.

systemBlock:
     "System"^ LPAREN! RPAREN! statements

;
```

A.3 Walker

```
{
     import util.*;
     import java.util.*;
     import antlr.debug.misc.ASTFrame;
}

class LogSimTreeWalker extends TreeParser;
```

```
options{
    importVocab = LogSim;
}

{
        LogSimSymTab globalSymTab = new LogSimSymTab();

     OperatorCollection opCollection = null;

     public OperatorCollection getOperatorCollection(){
          return opCollection;
     }

     LogSimSystem system = null;
     public void addOp(Operator Op){
          if(this.system == null){
               throw new LogSimException("Cannot add
operators until system is defined");
          }
          this.system.addOp(Op);
     }

     public HashMap getSignals(){
          return this.system.getSubTable().getISubTab();
     }
}

file
{
     //ASTFrame frame = new ASTFrame("Complete Tree", _t);
     //frame.setVisible(true);
}
     : #(FILE
          (#(COMPONENTS (component)*))?
          (#("System" system))?
        )
;

component
{
int ocount =0;
int icount =0;
LogSimComponent aComponent = null;
AST t = _t;

}
```

```
     :    #("Component"
          {
               //System.out.println("Entered Component");
          }
          #(i:IDENTIFIER
          {
               //System.out.println("Its name is
"+i.getText());
               aComponent = new
LogSimComponent(i.getText());
               if(globalSymTab.lookupComponent(i.getText())
!= null){
                    throw new LogSimException( i.getLine()
+ ":" + i.getColumn() + "Component  "+i.getText() + "
already exists");
               }
               else {
                    globalSymTab.addComponent(i.getText(),
aComponent);
               }
               aComponent.setTree(t);
          })
          #("In" icount = in[aComponent]
          {
                     aComponent.setInputCount(icount);
          }
          )
          #("Out" ocount = out[aComponent]
          {
               //System.out.println("Its output count is
"+ocount);
               aComponent.setOutputCount(ocount);
                  }
          )
          //#(STATEMENTS statements[aComponent] )
           )
;

in[LogSimComponent tComponent] returns [int icount]
{
 int count = 0;
 icount = 0;
 LogSimSubTable symTab = tComponent.getSubTable();
 LogSimVar var;
}
     :     (#(i:IDENTIFIER
          {
```

```
                //System.out.println("Symbol table right now
contains: "+ symTab + " Searching for " + i.getText());
                if (symTab.get (i.getText ())== null)
                {
                          var = symTab.create(i.getText());
                     var.setDef(true);
                }
                else{
                     throw new
LogSimException(i.getLine()+":"+i.getColumn() + i.getText()
+ " value already present");
                }
                //System.out.println("Input signal "+
i.getText() + " found");
                count++;
           }
            ))*
               {
                icount = count;
                //System.out.println("Icount" + icount);
               }
;

out[LogSimComponent tComponent] returns [int ocount]
{
 int count = 0;
 ocount = 0;
 LogSimSubTable symTab = tComponent.getSubTable();
 LogSimVar var;
}
        :     (#(i:IDENTIFIER
                {
                //System.out.println("Symbol table right now
contains: "+ symTab + " Searching for " + i.getText() +
i.getLine()+":"+i.getColumn());
                          if(symTab.get(i.getText())== null){
                     var = symTab.create(i.getText());
                     var.setDef(true);
                }
                          //else
                          //throw new
LogSimException(i.getLine() +":"+ i.getColumn() + " value
already present" + i.getText() );
                           //System.out.println("Output signal "+
i.getText() + " found");
                  count++;
                 }
```

```
            ))*
          {
               ocount = count;
               //System.out.println("Ocount" + ocount);
          }
;

componentActual[LogSimComponent parentComponent, ArrayList
actualParamsList, ArrayList actualOutputsList]
{
     LogSimSubTable symTab = parentComponent.getSubTable();
}

        :    #("Component"
                #(i:IDENTIFIER {})
                #("In" inArgsActual[symTab,
actualParamsList] )
          #("Out" outArgsActual[symTab, actualOutputsList]
)
                #(STATEMENTS statements[parentComponent] )
              )
        {
              // LogSimSubTable symTab =
parentComponent.getSubTable();
               //System.out.println("symtab:"+symTab);
               symTab.validate();
               //System.out.println("validated fine");
        }

;

inArgsActual[LogSimSubTable symTab, ArrayList
actualParamsList]
{
 //System.out.println("actualParams:"+actualParamsList);
 LogSimVar var = null;
 int index = 0;
 String formalParam;
}
        :
        (#(i:IDENTIFIER
                        {
                    //System.out.println("Found input Arg
"+ i.getText());
                    formalParam = i.getText();
                    var
=(LogSimVar)actualParamsList.get(index);
```

```
                    symTab.create(formalParam,var);
                    var.setDef(true);
               }

          {
               index++;
          }
         ))*

;

outArgsActual[LogSimSubTable symTab, ArrayList
actualOutputsList]
{
 //System.out.println("actualOutputs:"+actualOutputsList);
 LogSimVar var = null;
 int index = 0;
 String formalOutput;
}
        :
        (#(i:IDENTIFIER
                        {
                                //System.out.println("Found
output Arg "+ i.getText());
                                formalOutput = i.getText();
                                var
=(LogSimVar)actualOutputsList.get(index);
                    symTab.create(formalOutput,var);
                    var.setDef(true);
                        }

                {
                        index++;
                }
         ))*

;

statements[LogSimComponent tComponent]
{
     LogSimVar e;
     LogSimSubTable symTab = tComponent.getSubTable();
        LogSimVar var1,var2;
     ArrayList actualParams;
     ArrayList actualOutputs;
}
     :
```

```
     (#(EQUALS (eid: IDENTIFIER (BITINDICATOR bit: INT |
SARR index: INT EARR )?) e=expr[tComponent]
          {

     //System.out.println("symtab:"+symTab+"lvalue"+
eid.getText());
               var2 = symTab.get(eid.getText());
               if(var2 == null){
                     var2 = symTab.create(eid.getText());
                     var2.setDef(true);
               }
               if(var2.getDef() == false)
               {
                    var2.setDef(true);
               }
               if(bit != null){
                    //System.out.println("\n\nbit:"+bit);

     var2.setBit(Integer.parseInt(bit.getText()));
                    //System.out.println("Got bits" +
var2.getBit());
                    bit = null;
               }
               //System.out.println(eid.getText() + " has
to be combin assigned " + e);
               EQUALS op = new EQUALS();
               op.setInput(e);
               op.setOutput(var2);
               this.addOp(op);
               if(index!=null)    {

     op.onlySetBit(Integer.parseInt(index.getText()));
               index = null;
               }

          }
     )
     |
     #(i:IDENTIFIER actualParams = inputArgs[tComponent]
actualOutputs = outputArgs[tComponent]
          {
               //System.out.println("Calling
"+i.getText());
               LogSimComponent aComponent =
globalSymTab.lookupComponent(i.getText());
               if(aComponent == null){
```

```
                    throw new LogSimException(i.getLine() +
":" + i.getColumn() + "Component  "+i.getText() + "
undefined");
               }
               // First get a new copy of the component
looked up
               aComponent = new
LogSimComponent(aComponent);
               AST t = aComponent.getTree();
               //System.out.println("Inputs: " +
actualParams + "\nOutputs: " + actualOutputs );
               if(aComponent.getInputCount() !=
actualParams.size()){
                    throw new
LogSimException(i.getLine()+":"+i.getColumn()+" number of
input arguments not equal");
               }

     //System.out.println("outputCount"+aComponent.getOutpu
tCount());
                         if(aComponent.getOutputCount()!=
actualOutputs.size())
                                   throw new
LogSimException(i.getLine()+":"+i.getColumn() +"number of
output arguments not equal");
               this.componentActual(t, aComponent,
actualParams, actualOutputs);

          }
     )
     |
     #(SEQUAL sid: IDENTIFIER (BITINDICATOR sbit:INT | SARR
ndex:INT EARR )? e=expr[tComponent]
          {
                    //System.out.println("symtab:"+symTab);
               var1 = symTab.get(sid.getText());
                         if (var1 == null){
                              var1 =
symTab.create(sid.getText() );
                    var1.setDef(true);
               }
               if(var1.getDef()==false)
                    var1.setDef(true);
               if(sbit!=null){

     var1.setBit(Integer.parseInt(sbit.getText()));
                    sbit = null;
```

```
                }
                //System.out.println("Found sequential
equals");
                   SEQUALS op = new SEQUALS();
                         op.setInput(e);
                         op.setOutput(var1);
                         this.addOp(op);
                if(ndex!=null){

     op.onlySetBit(Integer.parseInt(ndex.getText()));
                ndex = null;
                }

             }
           ))*
;

inputArgs[LogSimComponent tComponent] returns [ArrayList
inputs]
{
 LogSimSubTable symTab = tComponent.getSubTable();
 inputs = new ArrayList();
 LogSimVar var = null;
}
           :
           #(INPUTARGS
                   ((i:IDENTIFIER
                         {
                //System.out.println("Found actual input Arg
"+ i.getText());
                         var = symTab.get(i.getText());
                         if(var == null)
                                 throw new LogSimException(
i.getLine()+":" + i.getColumn()+"input
arg"+(inputs.size()+1)+" not defined");
                         }
                    |intV:INT
                         {
                                 //System.out.println("Found
vector " + v);
                       var = new LogSimVar("temp");

var.setValue(LogSimAtom.generate(intV.getText()));

                         }
             |b:BIN
                  {
```

```
                    var = new LogSimVar("temp");

var.setValue(LogSimAtom.generate(b.getText()));
                }
                 )
                 {
                         inputs.add(var);
                 }
                 )*
        )
;

outputArgs[LogSimComponent tComponent] returns [ArrayList
outputs]
{//System.out.println("Found output args");
 int args =0;
 outputs = new ArrayList();
 LogSimSubTable symTab = tComponent.getSubTable();
 LogSimVar var = null;
}
     :
     #(OUTPUTARGS
          (i:IDENTIFIER
               {//System.out.println("Found output Arg " +
i.getText());
                var = symTab.get(i.getText());
                if(var == null){
                    var = symTab.create(i.getText());
                    var.setDef(true);
                }
                outputs.add(var);
               }
          )*
     )
     ;

system
{
     system = new LogSimSystem();
}
     :
     #(STATEMENTS statements[system])
     {
           LogSimSubTable symTab = system.getSubTable();
           //System.out.println("symtab:"+ symTab);
           symTab.validate();
           opCollection = system.getOperatorCollection();
```

```
          //System.out.println("Global Symbol table is: "+
system.getSubTable());
     }
        ;


expr[LogSimComponent tComponent] returns [LogSimVar var]
{
   LogSimVar v1,v2;
   LogSimSubTable symTab = tComponent.getSubTable();
   var = null;
}
     :    #(XOR v1 = expr[tComponent]  v2 =
expr[tComponent])
          {
               //System.out.println("Found xor");
               XOR op= new XOR();
                     op.setInput1(v1);
                        op.setInput2(v2);
                        var = new LogSimVar("temp");
                        op.setOutput(var);
               this.addOp(op);
          }
          |    #(OR v1 = expr[tComponent]  v2 =
expr[tComponent])
          {
               //System.out.println("Found OR");
               OR op =new OR();
                        op.setInput1(v1);
               op.setInput2(v2);
               var = new LogSimVar("temp");
                        op.setOutput(var);
               this.addOp(op);
          }
     |    #(AND v1 = expr[tComponent] v2 =
expr[tComponent])
          {
               //System.out.println("Found AND");
               AND op =new AND();
                        op.setInput1(v1);
               op.setInput2(v2);
               var = new LogSimVar("temp");
                        op.setOutput(var);
               this.addOp(op);
          }
     |    #(NOT v1 = expr[tComponent])
          {
```

```
                //System.out.println("Found NOT");
                NOT op =new NOT();
                op.setInput(v1);
                var = new LogSimVar("temp");
                op.setOutput(var);
                this.addOp(op);
          }
        |   #(i: INT
                 {
                       //  r =
LogSimAtom.generate(i.getText());
                //System.out.println("Returning "+r);
                var = new LogSimVar("temp");

     var.setValue(LogSimAtom.generate(i.getText()));
                 }
          )
     |   #(b: BIN
          {
                var = new LogSimVar("temp");

     var.setValue(LogSimAtom.generate(b.getText()));
          }
          )
     |  (intV: IDENTIFIER ( SARR start: INT (end: INT )?
EARR )?
          {
                //System.out.println("Found " +
i.getText());
                var = symTab.get(intV.getText());
                if(var == null){
                     var = symTab.create(intV.getText());
                }
                if(start != null){
                     //System.out.println("\n\n\n\nCreating
slice");
                          SLICE op = new SLICE();
                                 if(end != null){

op.setStartIndex(Integer.parseInt(start.getText()));

op.setEndIndex(Integer.parseInt(end.getText()));
                                 }
                     else {

     op.setStartIndex(Integer.parseInt(start.getText()));
```

```
     op.setEndIndex(Integer.parseInt(start.getText()));
                    }
                    op.setInput(var);
                         var = new LogSimVar("temp");
                    op.setOutput(var);
                    //System.out.println("Adding "+op);
                    this.addOp(op);
                        }
          }
             )
          ;
```

A.4 Utility classes

A.4.1 OperatorCollection.java

```
package util;

import java.util.*;

/**
 *
 *@auhor Mukul Khajanchi mk2603
 */
public class OperatorCollection {
      Vector operators;

      public OperatorCollection(){
            this.operators = new Vector();
      }

      public void addOperator(Operator pOp){
            //Vector inputs = pOp.getInputs();
            LogSimVar output = pOp.getOutput();

            Operator currentOp;
            if(!(pOp instanceof SEQUALS)){

      if(pOp.getInputs().contains(pOp.getOutput())){
                        throw new LogSimException("Cycle
found for "+pOp);
                  }
            }
            //System.out.println("pOp: " + pOp);
            for(int i = 0 ; i < operators.size() ; i++){
```

```
                    currentOp =
(Operator)this.operators.get(i);
                    //System.out.println("currentOp: " +
currentOp);
                    // If any of the outputs of this
Operator appear as Inputs to some other Operator
                       // already in the collection, then
add this Operator as that one's child
                    if(! (currentOp instanceof SEQUALS)){
                          //System.out.println("First
Condition");

      //System.out.println("currentOp.getInputs()"+
currentOp.getInputs());

      //System.out.println("pOp.getOutput()"+pOp.getOutput(
));

      //System.out.println("indexOf"+currentOp.getInputs().
indexOf(new LogSimVar("A")));

      if(currentOp.getInputs().contains(pOp.getOutput())){
                                if(areCompatible(pOp,
currentOp)){

      //System.out.println(currentOp+" has input same as
the output of " + pOp);
                                      // currentOp depends
on pOp. Hence, add oOp to the children of currentOp

      currentOp.addChild(pOp);
                                }
                          }
                    }

                    // If any of the inputs to this Operator
appear as the output to some other Operator
                    // already in the collection, then add
that Operator, this one's child
                    if(! (pOp instanceof SEQUALS)){
                          //System.out.println("Second
condition");

      //System.out.println("pOp.getInputs()"+
pOp.getInputs());
```

```
//System.out.println("current.getOutput()"+currentOp.getOut
put());

      if(pOp.getInputs().contains(currentOp.getOutput())){
                              if(areCompatible(currentOp,
pOp)){

      //System.out.println(currentOp + " has output same as
one of the inputs of "+pOp);
                                    //pOp depends on
currentOp. Hence add currentOp to the children of pOp

      pOp.addChild(currentOp);
                              }
                        }
                  }

                  // If any of the outputs of this
operator is same as the output of any other operator
                  // throw an error. But this condition is
only valid if it is not a SEQUALS node
                  if(! (pOp instanceof SEQUALS)){

      if(pOp.getOutput().equals(currentOp.getOutput())){
                              //throw new
LogSimException(currentOp.getOutput() + " appears as output
in more than two operators!");
                        }
                  }
            }

            operators.add(pOp);
      }

        public List topoSort(){
                List sortedList = new ArrayList();
                Node node;
                for(int i = 0 ; i < this.operators.size() ;
i++){
                        node = (Node)this.operators.get(i);
                        if(node.isWhite()){
                                node.DFS(sortedList);
                        }
                }
                return sortedList;
        }
```

```
	public boolean areCompatible(Operator pEquals,
Operator pSlice){
		if(!(pEquals instanceof EQUALS) || !(pSlice
instanceof SLICE)){
			return true;
		}
		else {
			SLICE mySlice = (SLICE)pSlice;
			EQUALS myEquals = (EQUALS)pEquals;
			if(myEquals.getOnlySetBit() > -1
				&& ((myEquals.getOnlySetBit() <
mySlice.getStartIndex())
					|| (myEquals.getOnlySetBit()
> mySlice.getEndIndex()))){
				return true;
			}
			else {
				return false;
			}
		}
	}

	public boolean areReferringDiffBitsOfSameVar(Operator
pOperator1, Operator pOperator2){
		if(pOperator1 instanceof SLICE){
			return true;
		}
		if(pOperator2 instanceof SLICE){
			return true;
		}
		if(pOperator1 instanceof EQUALS && pOperator2
instanceof SLICE){
			EQUALS pEQUALS = (EQUALS)pOperator1;
			SLICE pSlice = (SLICE)pOperator2;
			if(!(pSlice.getStartIndex() >
pEQUALS.getOnlySetBit()
				&& pEQUALS.getOnlySetBit() >
pSlice.getEndIndex())){
				return true;
			}
		}
		return false;
	}

	public String toString(){
		String opColl = "";
```

```
            Operator op;
            Operator child;
            for(int i = 0 ; i < this.operators.size() ;
i++){
                  op = (Operator)this.operators.get(i);
                  opColl += "\n" + op + " depends on: ";
                  for(int j = 0 ; j <
op.getChildren().size() ; j++){
                        child =
(Operator)op.getChildren().get(j);
                        opColl += child + " ";
                  }
            }
            return opColl;
      }
}
```

A.4.2 Operator.java

```
package util;
import java.util.*;

/**
 *
 *@auhor Mukul Khajanchi mk2603
 */
public abstract class Operator extends Node{
      private LogSimVar output;

      public LogSimVar getOutput(){
            return this.output;
      }

      public void setOutput(LogSimVar pOutput){
            this.output = pOutput;
      }

      public LogSimAtom getValue(){
            return this.getOutput().getValue();
      }

      public void setValue(LogSimAtom pAtom){
            this.output.setValue(pAtom);
      }

      public abstract void evaluate() throws
LogSimException;
```

```
	public abstract Vector getInputs();
	public abstract String toString();
}
```

A.4.3 UnOperator.java

```
package util;

import java.util.*;

/**
 *
 *@auhor Mukul Khajanchi mk2603
 */
public abstract class UnOperator extends Operator{
	protected LogSimVar input;

	public void setInput(LogSimVar pInput){
		this.input = pInput;
	}

	public Vector getInputs(){
		Vector inputVector = new Vector();
		inputVector.add(this.input);
		return inputVector;
	}
}
```

A.4.4 BinOperator.java

```
package util;
import java.util.*;

public abstract class BinOperator extends Operator{
     protected LogSimVar input1;
     protected LogSimVar input2;

     public void setInput1(LogSimVar pInput1){
          this.input1 = pInput1;
     }

     public void setInput2(LogSimVar pInput2){
                this.input2 = pInput2;
        }

     public Vector getInputs(){
          Vector inputVector = new Vector();
```

```
            inputVector.add(this.input1);
            inputVector.add(this.input2);
            return inputVector;
      }
}
```

A.4.5 SEQUALS.java

```
package util;

/**
 *
 *@auhor Mukul Khajanchi mk2603
 */
public class SEQUALS extends UnOperator{
      // Although it might appear that a Sequential
Assignment node gets multiple inputs,
      // finally it just consists of eiher a binary
expression or a unary expression,
      // which is going to constitute one atom only
      private LogSimVar newV;
      private int onlySetBit =-1;

      public void setNew(LogSimVar pNew){
            this.newV = pNew;
      }

      public LogSimVar getNew(){
            return this.newV;
      }

      public void evaluate(){
            // By default set the value as zero
            // because we want to have an output for every
sequential node
            if(super.getValue() == null){

      super.setValue(LogSimAtom.generate("0"));
            }

               if(this.onlySetBit > -1){
                        try{

this.getNew().getValue().setBit(onlySetBit,
input.getValue());
                        }
                        catch(LogSimException lse){
```

```
                                    throw new
LogSimException(lse.getMessage() + " for " +
super.getOutput());
                              }
                  }

                  //System.out.println("Evaluated and have set
value to "+ super.getValue() + " and new value to " +
this.getNew());
            }

            public void setInput(LogSimVar pVar){
                  this.setNew(pVar);
            }

            public void update(){
                  if(this.getNew().getValue() == null){

            this.getNew().setValue(LogSimAtom.generate("0"));
                  }
                  //System.out.println("this.getNew(): " +
this.getNew() + " .getValue(): " +
this.getNew().getValue());
                  super.setValue(this.getNew().getValue());
                  //System.out.println("Updated and have set
value to "+super.getValue());
            }

            public SEQUALS(){
                  // Set the input as zero by default
                  //LogSimVar zero = new LogSimVar("zero");
                  //zero.setValue(LogSimAtom.generate("0"));
                  //super.setOutput(zero);
            }

            public void onlySetBit(int pOnlySetBit){
                  this.onlySetBit = pOnlySetBit;
            }

            public String toString(){
                  // This is an exceptional case where the input
is stored in the "new" attribute
                     return super.getOutput() + " := " +
this.getNew();
              }
}
```

A.4.7 LogSimVar.java

```
package util;

/**
 *
 *@auhor Mukul Khajanchi mk2603
 */
public class LogSimException extends Error {
     public LogSimException(String message){
          super(message);
     }
}
```

A.4.8 LogSimAtom.java

```
package util;
import java.util.Vector;

/**
 *
 *@auhor Mukul Khajanchi mk2603
 */
public class LogSimAtom {
     private Vector vector = new Vector();

     public Vector getVector(){
          return this.vector;
     }

     public int size(){
          return this.vector.size();
     }

     public void add(int elem){
          this.vector.add(new Integer(elem));
     }

     private int get(int index) {
          if(index < 0){
               throw new LogSimException("Trying to access
a bit with an index less than 0");
          }
          else if (index > this.size()){
               throw new LogSimException("Trying to access
a bit with an index greater than the size of the vector");
          }
```

```
          else {
               return
((Integer)(this.vector.get(index))).intValue();
          }
     }

     private LogSimAtom(){
     }

     private LogSimAtom(String tokenText){
          this(tokenText, false);
     }

     public static LogSimAtom generate(String tokenText){
          if(tokenText.startsWith("^")){
               return new LogSimAtom(tokenText.substring(1,
tokenText.length()), true);
          }
          else {
               return new LogSimAtom(tokenText);
          }
     }

     private LogSimAtom(String tokenText, boolean
isBinary){
          if(!isBinary){
               tokenText =
Integer.toBinaryString(Integer.parseInt(tokenText));
          }
                StringBuffer sb = new
StringBuffer(tokenText);
                for(int i = 0 ; i < sb.length() ; i++){
                    this.add(Character.digit(sb.charAt(i),
10));
                }

     }

     private LogSimAtom(LogSimAtom pAtom, int pSize) {
          if(pSize > pAtom.size()){
               int valToAdd = 0;
               // In the special case that pAtom's size is
just 1, we pad the entire Vector with the value of
               // it one and only bit
               if(pAtom.size() == 1){
                    valToAdd = pAtom.get(0);
               }
```

```
                for(int i = pSize ; i > pAtom.size() ; i--
 ){
                     this.add(valToAdd);
                }
          }
          for(int i = 0 ; i< pAtom.size() ; i++){

                this.add(pAtom.get(i));
          }
     }

     public LogSimAtom(LogSimAtom pAtom){
          this.vector = new Vector(pAtom.getVector());
     }

     public LogSimAtom and(LogSimAtom pAtom){
          int currSize = this.size();
          int paramSize = pAtom.size();
         int resultSize = currSize > paramSize ? currSize :
paramSize;
          LogSimAtom normalizedAtom = new LogSimAtom(this,
resultSize);
         LogSimAtom normalizedPAtom = new LogSimAtom(pAtom,
resultSize);

          return
normalizedAtom.andNormalized(normalizedPAtom);

     }

     private LogSimAtom andNormalized(LogSimAtom pAtom){
          LogSimAtom resultAtom = new LogSimAtom();
          for(int i = 0 ; i < this.size() ; i++){
                resultAtom.add((this.get(i)) &
(pAtom.get(i)));
          }
          return resultAtom;
     }

        public LogSimAtom or(LogSimAtom pAtom){
                int currSize = this.size();
                int paramSize = pAtom.size();
                int resultSize = currSize > paramSize ?
currSize : paramSize;
                LogSimAtom normalizedAtom = new
LogSimAtom(this, resultSize);
```

```
                LogSimAtom normalizedPAtom = new
LogSimAtom(pAtom, resultSize);

                return
normalizedAtom.orNormalized(normalizedPAtom);

        }

        private LogSimAtom orNormalized(LogSimAtom pAtom){
                LogSimAtom resultAtom = new LogSimAtom();
                for(int i = 0 ; i < this.size() ; i++){
                        resultAtom.add((this.get(i)) |
(pAtom.get(i)));
                }
                return resultAtom;
        }

        public LogSimAtom xor(LogSimAtom pAtom){
                int currSize = this.size();
                int paramSize = pAtom.size();
                int resultSize = currSize > paramSize ?
currSize : paramSize;
                LogSimAtom normalizedAtom = new
LogSimAtom(this, resultSize);
                LogSimAtom normalizedPAtom = new
LogSimAtom(pAtom, resultSize);

                return
normalizedAtom.xorNormalized(normalizedPAtom);

        }

        private LogSimAtom xorNormalized(LogSimAtom pAtom){
                LogSimAtom resultAtom = new LogSimAtom();
                for(int i = 0 ; i < this.size() ; i++){
                        resultAtom.add((this.get(i)) ^
(pAtom.get(i)));
                }
                return resultAtom;
        }

        public LogSimAtom not(){
                int currSize = this.size();
                LogSimAtom resultAtom = new LogSimAtom();
                for(int i = 0 ; i < this.size() ; i++){
                        resultAtom.add(1 - this.get(i));
                }
```

```
                return resultAtom;

        }

     public LogSimAtom slice(int start, int end){
          if(end < start){
               throw new LogSimException("End Index for
slicing is less than Start Index");
          }
          else if(start < 0){
               throw new LogSimException("Vector size is
"+this.size() + " trying to slice[0...");
          }
          else if(!(end < this.size())){
               throw new LogSimException("Vector size is "
+ this.size() + " trying to slice ..." + end+ "]");
          }
          else {
               LogSimAtom result = new LogSimAtom();
               // User's start is the bits location
starting from the RHS
               for(int i = end ; i >= start ; i--){
                    int actualIndex = this.size() - i - 1;
                    result.add(this.get(actualIndex));
               }
               //System.out.println("Sliced and returning
"+result);
               return result;
          }
     }

     public LogSimAtom getOnlyBits(int numberOfBits){
          LogSimAtom result = new LogSimAtom();
          if(numberOfBits > this.size()){
               int numberOfBitsToPad = numberOfBits -
this.size();
               //System.out.println("numberOfBitsToPad:
"+numberOfBitsToPad);
               for(int i = 0 ; i < numberOfBitsToPad ;
i++){
                    result.add(0);
               }

     //System.out.println("numberOfBits"+numberOfBits);
               for(int i = 0 ; i < this.size() ; i++){
                    result.add(this.get(i));
```

```
                }
            }
            else if(numberOfBits < this.size()){
                for(int i = numberOfBits ; i < this.size() ;
i++){
                    result.add(this.get(i));
                }
            }
            else {
                result = new LogSimAtom(this);
            }
            return result;
        }

        public String toString(){
            String str = "";
            for(int i = 0 ; i < this.size(); i++){
                str += this.get(i) + " ";
            }
            return str;
        }

        public int getBit(int bitIndex){
            int indexInVector = this.size() - bitIndex - 1;
            return this.get(indexInVector);
        }

        public void setBit(int bitIndex, LogSimAtom pValue){
            int indexInVector = this.size() - bitIndex - 1;
            //System.out.println("bitIndex: "+ bitIndex);
                 //System.out.println("indexInVector: "+
indexInVector);

            if(indexInVector < 0){
                // This means it is trying to set a bit
which is not yet accessed
                // So we will pad the rest of the bits with
0
                // and add the new bit
                //System.out.println("Initially vector was
"+this.vector);
                this.vector = (new LogSimAtom(this, bitIndex
+ 1)).getVector();
                //System.out.println("Made it as:
"+this.vector);
                this.vector.setElementAt(pValue.getBit(0),
0);
```

```
                //System.out.println("Added the element and
now the vector is "+this.vector);
                //throw new LogSimException("Cannot access
the " + bitIndex + "th signal ");
           }
           else {
                this.vector.setElementAt(pValue.getBit(0),
indexInVector);
           }
     }


     public static void main(String[] args){
           /*
          LogSimAtom myAtom1 = new LogSimAtom("0110", true);
           System.out.println("myAtom1"+ myAtom1);
           LogSimAtom myAtom2 = new LogSimAtom("18", false);
           System.out.println("myAtom2 "+ myAtom2);
           System.out.println("myAtom1 and myAtom2 "+
myAtom1.and(myAtom2));
           System.out.println("myAtom1 or myAtom2 "+
myAtom1.or(myAtom2));
           System.out.println("myAtom1 xor myAtom2 "+
myAtom1.xor(myAtom2));
           System.out.println("!myAtom1 "+ myAtom1.not());
           System.out.println("!myAtom2 "+ myAtom2.not());
           System.out.println("myAtom1 NAND myAtom2 " +
(myAtom1.and(myAtom2)).not());
           System.out.println("myAtom2 "+myAtom2);
           System.out.println("myAtom2[0..4] " +
myAtom2.slice(0,4));
           System.out.println("myAtom2[0..3] " +
myAtom2.slice(0,3));
           System.out.println("myAtom2[0..1] " +
myAtom2.slice(0,1));
           System.out.println("myAtom2[0..0] " +
myAtom2.slice(0,0));
           System.out.println("myAtom2[1..1] " +
myAtom2.slice(1,1));
           System.out.println("myAtom2[2..2] " +
myAtom2.slice(2,2));
           System.out.println("myAtom2[3..3] " +
myAtom2.slice(3,3));
           System.out.println("myAtom2[4..4] " +
myAtom2.slice(4,4));
           System.out.println("myAtom2[5..5] " +
myAtom2.slice(5,5));
```

```
            LogSimAtom myAtom3= new LogSimAtom("1",true);
                 System.out.println("myAtom3 "+myAtom3);
                   LogSimAtom myAtom4 = new
LogSimAtom("12");
                   System.out.println("myAtom4: " +
myAtom4);
          System.out.println("myAtom4.getOnlyBits(8):
"+myAtom4.getOnlyBits(8));
          System.out.println("myAtom4.getOnlyBits(2):
"+myAtom4.getOnlyBits(2));
          */
          LogSimAtom a = LogSimAtom.generate("^1010");
          LogSimAtom control0 = LogSimAtom.generate("^0");
          LogSimAtom control1 = LogSimAtom.generate("^0");
          System.out.println("not(control0) " +
control0.not());
          System.out.println("not(control1) " +
control1.not());
          LogSimAtom notAnded =
control0.not().and(control1.not());
          System.out.println("notAnded: "+ notAnded);
          System.out.println("a.and(notAnded) "+
a.and(notAnded));
     }
}
```

A.4.9 AND.java

```
package util;
public class AND extends BinOperator{

     public void evaluate(){

     super.setValue(input1.getValue().and(input2.getValue()
));
     }

     public String toString(){

          return super.getOutput() + " = " + input1 + " AND
" + input2;
     }
}
```

A.4.10 EQUALS.java

```
package util;
```

```
public class EQUALS extends UnOperator{
     // Although it might appear that a Combinational
Assignment node gets multiple inputs,
     // finally it just consists of eiher a binary
expression or a unary expression,
     // which is going to constitute one atom only

     private int onlySetBit = -1;
     public void evaluate(){
          if(this.onlySetBit > -1){
               if(super.getOutput().getValue() == null){
                    // If output not yet declared,
initialize it

     super.getOutput().setValue(LogSimAtom.generate("0"));
               }
               try{

     super.getOutput().getValue().setBit(onlySetBit,
input.getValue());
               }
               catch(LogSimException lse){
                    throw new
LogSimException(lse.getMessage() + " for " +
super.getOutput());
               }
          }
          else {
               super.setValue(input.getValue());
          }
     }

     public String toString(){
                return super.getOutput() + " = " + input;
        }

     public void onlySetBit(int pOnlySetBit){
          this.onlySetBit = pOnlySetBit;
     }

     public int getOnlySetBit(){
          return this.onlySetBit;
     }
}
```

A.4.3 LogSimComponent.java

```
package util;
import antlr.collections.AST;
/**
 *
 *@auhor Mukul Khajanchi mk2603
 */
public class LogSimComponent {
     private String name;
        private int icount;
        private int ocount;
        private LogSimSubTable symTab;
     private AST tree;

     public String getName(){
          return this.name;
     }

     public void setName(String pName){
          this.name = pName;
     }

     public void setTree(AST tree){
          this.tree = tree;
     }

     public AST getTree(){
          return (AST)this.tree;
     }

     public LogSimComponent(String pName){
          this.symTab = new LogSimSubTable();
          this.setName(pName);
     }

     public LogSimComponent(LogSimComponent pComponent){
          this.setName(pComponent.getName());
          this.symTab = new LogSimSubTable();
          this.setTree(pComponent.getTree());
          this.setInputCount(pComponent.getInputCount());
          this.setOutputCount(pComponent.getOutputCount());
     }

     public void setInputCount(int inCount){
          this.icount = inCount;
     }
```

```
     public void setOutputCount(int outCount){
          this.ocount = outCount;
     }

     public int getInputCount(){
                return this.icount;
        }

        public int getOutputCount(){
                return this.ocount;
        }

        public void check(LogSimAtom ch){
     }

     public LogSimSubTable getSubTable(){
          return this.symTab;
     }

}
```

A.4.11 LogSimSubTable.java

```
package util;
import java.io.InputStream;
import java.io.Reader;
import java.util.*;

/*
 *
 * @author Nithya Ganesan
 */
public class LogSimSubTable{

     private HashMap iSubTab;
     public LogSimSubTable(){
          iSubTab = new HashMap();
     }

       public LogSimVar create(String name){
          LogSimVar var = new LogSimVar (name);
          this.iSubTab.put(name, var);
          return var;
     }


     public LogSimVar get(String name){
```

```
                return (LogSimVar)this.iSubTab.get(name);


     }

     public void create(String pName, LogSimVar pVar){
          this.iSubTab.put(pName, pVar);
     }

     public void validate(){
          Set s =iSubTab.keySet();
          for (Iterator it=s.iterator(); it.hasNext(); ) {
          String key =(String)it.next();
          LogSimVar var = (LogSimVar)iSubTab.get(key);
          if(var.getDef()==true)
          continue;
          throw new LogSimException( key +" not defined" );
                }
     }

     public HashMap getISubTab(){
          return this.iSubTab;
     }

     public String toString(){
          return this.iSubTab.toString();
     }
}
```

A.4.12 LogSimSystem.java

```
package util;

/**
 *
 *@author Mukul Khajanchi mk2603
 */
public class LogSimSystem extends LogSimComponent {

     public LogSimSystem(){
          super("SYSTEM");
          ops = new OperatorCollection();
     }
        private OperatorCollection ops;

        public void addOp(Operator pOp){
                this.ops.addOperator(pOp);
```

```
        }

        public OperatorCollection getOperatorCollection(){
                return this.ops;
        }

}
```

A.4.13 LogSimException.java

```
package util;

/**
 *
 *@auhor Mukul Khajanchi mk2603
 */
public class LogSimException extends Error {
     public LogSimException(String message){
          super(message);
     }
}
```

A.4.14 LogSimSymTab.java

```
package util;

/**
 *
 *@auhor Mukul Khajanchi mk2603
 */
public class LogSimException extends Error {
     public LogSimException(String message){
          super(message);
     }
}
```

A.4.15 Or.java

```
package util;

/**
 *
 *@auhor Mukul Khajanchi mk2603
 */
public class OR extends BinOperator{

       public void evaluate(){
```

```
            //System.out.println("\n\nIn OR");
            //System.out.println("input1:
"+input1.getValue());
            //System.out.println("input2:
"+input2.getValue());

      super.setValue(input1.getValue().or(input2.getValue()
));
      }

      public String toString(){
            return super.getOutput() + " = " + input1 + "
OR " + input2;
      }
}
```

A.4.16 SLICE.java

```
package util;
/**
 *
 *@auhor Mukul Khajanchi mk2603
 */
public class SLICE extends UnOperator{

      private int startIndex;
      private int endIndex;

      public void evaluate(){
            if(input.getValue() == null){
                  throw new LogSimException("Trying to
slice a Vector which is not defined");
            }

      super.setValue(input.getValue().slice(this.getStartIn
dex(), this.getEndIndex()));
      }

        public String toString(){
                return super.getOutput() + " = " + input +
"[" + this.getStartIndex() + "..." + this.getEndIndex() +
"]";
        }

      public void setStartIndex(int pStartIndex){
            this.startIndex = pStartIndex;
      }
```

```
	public void setEndIndex(int pEndIndex){
		this.endIndex = pEndIndex;
	}

	public int getStartIndex(){
		return this.startIndex;
	}

	public int getEndIndex(){
		return this.endIndex;
	}
}
```

A.4.17 XOR.java

```
package util;

/**
 *
 *@auhor Mukul Khajanchi mk2603
 */
public class XOR extends BinOperator{

	public void evaluate(){

	super.setValue(input1.getValue().xor(input2.getValue(
)));
	}

	public String toString(){
		  return super.getOutput() + " = " + input1 +
" XOR " + input2;
		}

}
```

A.4.18 NOT.java

```
package util;

/**
 *
 *@auhor Mukul Khajanchi mk2603
 */
public class NOT extends UnOperator{
	public void evaluate(){
```

```
            super.setValue(input.getValue().not());
            //System.out.println("Evaluated and set output
as "+ super.getValue());
      }

        public String toString(){
                return super.getOutput() + " = NOT " +
input;
        }

}
```

A.4.19 NODE.java

```
package util;

import java.util.*;

/**
 *
 *@author Mukul Khajanchi mk2603
 */
public class Node{

      public static int WHITE = 0;
      public static int GREY = 1;
      public static int BLACK = 2;

      private int color;
      private Vector children;

      public Node(){
            this.children = new Vector();
      }

      public Node(Object pObject){
            this.children = new Vector();
            //this.setValue(pObject);
      }

      public int getColor(){
            return this.color;
      }

      public void setColor(int pColor){
            this.color = pColor;
      }
```

```
	public Vector getChildren(){
		return this.children;
	}

	public void addChild(Node pNode){
		this.getChildren().add(pNode);
	}

	public boolean isWhite(){
		return this.color == Node.WHITE;
	}

	public boolean isGrey(){
		return this.color == Node.GREY;
	}

	public boolean isBlack(){
		return this.color == Node.BLACK;
	}

	public void setWhite(){
		this.setColor(Node.WHITE);
	}

	public void setGrey(){
		this.setColor(Node.GREY);
	}

	public void setBlack(){
		this.setColor(Node.BLACK);
	}

	public Node DFS(List pList){
		//System.out.println("Now visiting "+ this);
		this.setGrey();
		Node dfsNode = new Node();
		Node child;
		for(int i = 0 ; i < this.getChildren().size() ;
i++){
			child = (Node)this.getChildren().get(i);
			//System.out.println("Found child " +
child + ".");
			if(child.isWhite()){

	dfsNode.addChild(child.DFS(pList));
			}
```

```
                  else if(child.isGrey()){
                        throw new LogSimException("Cycle
of combinational nodes detected for "+ this + " and " +
child);
                        //System.out.println("Found a
cycle!");
                  }
            }
            this.setBlack();
            //System.out.println("Processed " + this);
            pList.add(this);
            return dfsNode;
      }
}
```

A.4 Driver Program

```
/**
 *
 * The driver class of LogSim which calls the scanner, parser,
 * walker and finally does a topological sort of the operators
 * and executes them for the number of cycles requested
 *
 * Adopted from the Main.java provided on the class homepage
 *
 * @author Mukul Khajanchi
 */
import java.io.*;
import java.util.*;
import util.*;

import antlr.CommonAST;
import antlr.collections.AST;
import antlr.debug.misc.ASTFrame;

public class Main {
      public static void main(String args[]) {
            try {
                        DataInputStream input = new
DataInputStream(System.in);

                        // Create the lexer and parser and feed them
the input
                  long startTime = System.currentTimeMillis();
                        LogSimLexer lexer = new LogSimLexer(input);
                  long endTime = System.currentTimeMillis();
                  //System.out.println("Scanned the file in " +
(endTime-startTime) + " milliseconds");
                  startTime = System.currentTimeMillis();
                        LogSimParser parser = new LogSimParser(lexer);
                  parser.setASTNodeClass("CommonASTWithLines");
```

```
                  parser.file(); // "file" is the main rule in the
parser
                  endTime = System.currentTimeMillis();
                  if(lexer.nr_error > 0 || parser.nr_error > 0){
                        System.err.println("Parsing errors found!
Stop");
                        return;
                  }

                  //System.out.println("Parsed in " + (endTime-
startTime) + " milliseconds");
                  // Get the AST from the parser
                        CommonAST parseTree =
(CommonASTWithLines)parser.getAST();

                        // Print the AST in a human-readable format
                        //System.out.println("parseTree: \n" +
parseTree.toStringList());

                  if(args.length > 0 && "-w".equals(args[0])){
                              // Open a window in which the AST is
displayed graphically
                              ASTFrame frame = new ASTFrame("AST from
the LogSim parser", parseTree);
                              frame.setVisible(true);
                  }
                  startTime = System.currentTimeMillis();
                  LogSimTreeWalker walker = new LogSimTreeWalker();
                  walker.file(parseTree);
                  endTime = System.currentTimeMillis();
                  //System.out.println("Walked in " + (endTime-
startTime) + " milliseconds");
                  OperatorCollection ops =
walker.getOperatorCollection();
                  startTime = System.currentTimeMillis();

                  //System.out.println("\n\n\n\nOperator Collection: "
+ ops);
                  List sortedOperators = ops.topoSort();
                  endTime = System.currentTimeMillis();

                  //System.out.println("Sorted in " + (endTime-
startTime) + " milliseconds");
                  //System.out.println("\n\n\n\nSorted Operators: " +
sortedOperators);

                  int numOfTick = 10;
                  int tickToStartPrintingFrom = 0;

                  if(args.length > 0){
                        if(args.length == 2){
                              try{
                                    tickToStartPrintingFrom =
Integer.parseInt(args[0]);
                                    numOfTick =
Integer.parseInt(args[1]);
                              }
```

```
                              catch(NumberFormatException nfe){
                                    System.err.println(usageHelp());
                                    return;
                              }
                        }
                        else if(args.length == 1){
                              try{
                                    numOfTick =
Integer.parseInt(args[0]);
                              }
                              catch(NumberFormatException nfe){

      System.err.println(usageHelp());
                                    return;
                                    }
                        }
                        else {
                              System.out.println(usageHelp());
                        }
                  }
                  startTime = System.currentTimeMillis();
                  Operator op;
                  for(int tick = 0 ; tick < numOfTick ; tick++){
                              for(int i = 0 ; i <
sortedOperators.size() ; i++){
                                    op =
(Operator)sortedOperators.get(i);
                                    //System.out.println("Evaluating
"+op);
                                    op.evaluate();
                              }
                        if(tick >= tickToStartPrintingFrom){
                              System.out.println("Tick "+tick + ": ");
                              // Print out all the signals in the
system
                              HashMap signalsHashMap =
walker.getSignals();
                              Iterator it =
signalsHashMap.keySet().iterator();
                              LogSimVar var;

                              while (it.hasNext()) {
                                    // Get key
                                    String name = (String)it.next();
                                    var =
(LogSimVar)signalsHashMap.get(name);
                                    System.out.print(name + " = " +
var.getValue() + "\t");

                              }
                        }

                        System.out.println("\n");
                        for(int i = 0 ; i < sortedOperators.size() ;
i++){
                              op =
(Operator)sortedOperators.get(i);
```

```
                              if(op instanceof SEQUALS){

      //System.out.println("Updating "+op);
                                          ((SEQUALS)op).update();

      //System.out.println(op.getOutput() + " is " + op.getValue());
                                    }
                        }

                  }
                  endTime = System.currentTimeMillis();
                  //System.out.println("Evaluated in " + (endTime-
startTime) + " milliseconds");
            }
            catch(LogSimException lse){
                  System.err.println("Error: " + lse.getMessage());
                  return;
            }
            catch(Exception e) {
                  System.err.println("Exception: "+e);
                  e.printStackTrace();
                  return;
            }
      }// End Main

      public static String usageHelp(){
            return "Usage:\n"
                  + "";
      }
}
```

A.5 The LogSim executable

```
#!/bin/bash
numArgs=$#
arg=$1
filename=$arg
dirname=`dirname $filename`
if [ ! -n "$1" ]
then
  echo "Usage: `basename $0` <filename>"
  exit 1
fi


case $1 in
  "--v" | "-version")
      echo "LogSim Version 1.0"
      exit 0
      ;;
  "--h" | "-help")
      echo "LogSim"
      exit 0
      ;;
esac
```

```
if test -f $filename
then
      found=true
else
      echo "Error: cannot find $filename"
      exit 1
fi

for file in `grep -i "Import*" $filename | awk '{print $2}'`
do
      if test -f $dirname/$file
      then
            cat $dirname/$file >> $filename.tmp
      else
            echo "Error: cannot import $file"
            exit 1
      fi
done
grep -v "Import*" $filename >> $filename.tmp

java Main < $filename.tmp $2 $3
rm $filename.tmp
exit 0
```