LCSL

Logic Circuit Simulation Language

Bogdan Caprita, Julian Maller, Sachin Nene, Chaue Shen

December 19, 2003

Chapter 1
Introduction

1.1 Introduction

The Logic Circuit Simulation Language (LCSL) has been built to design logical circuits in a new, intuitive way. The classical approach to building circuits has always been using declarative hardware design languages, mainly because the building blocks used in these declarative programs are how the circuits are actually mapped out in the hardware. But on a conceptual level, human beings think of circuits in a different way; that of a web of functions that have inputs and outputs.

This is where LCSL comes in. With its functional programming philosophy, LCSL allows users to design circuits at a highly conceptual level. LCSL then transforms the code into the code of the popular declarative HDL Verilog so that the circuit can be simulated and synthesized. By producing an extra layer of abstraction, LCSL frees the user from worrying about the nitty-gritty of long, convoluted code of Verilog that is necessary for the actual production of the circuit.

1.2 Background

The logic circuit is an integral part of the transistors that are in virtually all aspects of computational electronics—from the airbags in a car to the alarm in a watch. The fundamental aspect of a logic circuit is that it operates on a digital signal that always carries one of only two values. The representation of these values varies through different technologies, whether it is a high or low voltage at a very low hardware level or Boolean representations of true and false in a high-level programming language.

Logic circuits act as a physical representation of a mathematical or logical function between a set of inputs and a set of outputs. Inputs are usual carried in by bit wires that carry a high or low voltage (which, again, can be translated to true/false values on a very high level). The physical medium that is a circuit transforms these signals in such a way as to produce a specific set of outputs that are sent out by bit wires as well. Combining several of these “function” circuits results in a useful tool for data, whether it be simple mathematical functions such

as adding and subtracting, or at a much more complex level, displaying graphics on a CRT monitor. The average microchip in a computer contains millions upon millions of these circuits forming the processing power of today's computing products.

A structural representation is always a key component of the design process of a digital logical circuit. More and more, software and simulation is becoming an essential part of most fields of experimentation. Circuit research is not an exception, and simulation through software before even touching a piece of silicon is vital in making the exercise more efficient and worthwhile. Designing a digital logic circuit on a computer and simulating it with various inputs allows electrical engineers to tweak and prod at their discretion without the expense of materials or loss of valuable time.

1.3 Related Work

The two hardware design languages that have ruled the circuit design landscape for the past twenty years have been Verilog and VHDL. Both languages implement all steps of the hardware design process: conceptual design, simulation, and synthesis. Conceptual design allows a user to create the circuit of his choice using the fundamental aspects of the specific HDL. Simulation allows the user to test the conceptual design with real inputs and outputs. Synthesis allows the user to create the actual net list of the circuit from the conceptual design.

Both languages are extremely powerful with respect to simulation and synthesis. The levels of optimization their compilers have produced are at this point unparalleled. This is the reason that LCSL leaves the work of simulation and synthesis to Verilog. The one weak spot that can be noted in Verilog (and especially VHDL) is that the conceptual design phase is not highly intuitive and can thus have a high learning curve.

In LCSL, we have taken conceptual design one step further than other HDLs. By implementing the benefits of functional programming, we have made it easier than ever to build circuits.

LCSL has based its syntax on another HDL called Confluence. Confluence, designed by Launchbird Design Systems, Inc., was the first to introduce the functional language programming interface to hardware design languages. We believed the syntax of Confluence is easy to understand and was the best choice for our language.

1.4 Goals

1.4.1 Simplicity

The main purpose of designing this language was to make it conceptually easier to design a circuit. LCSL translates its code into comparable Verilog that can be used for simulation and synthesis. By focusing specifically on the conceptual design, we made our code less verbose and easier to understand than Verilog.

1.4.2 Portability

LCSL is developed through the Java programming language, and implicitly, via the Java Virtual Machine. Because hardware-specific JVMs have been developed for almost all platforms, Java has become famous for being a highly portable language. Because LCSL is based on Java, it inherits the quality of portability.

1.4.3 Modularity

Like other hardware design languages, LCSL maintains the notion of being able to define circuits as black boxes. Different modules can be defined encompassing one or more circuits or even one or more other modules. This type of compartmentalization resembles object-oriented programming where a user doesn't have to know how a module works but what inputs it needs and what outputs it gives.

1.4.4 Flexibility

With LCSL, you no longer has to be bound to the limits of programming sequentially. LCSL gives you quite a bit of room to work under since there is no artificial constraint of requiring wires to be connected in a certain order. Wires can also be left unconnected and optionally connected inside the component itself (like input wires).

1.4.5 Error Avoidance

With a functional programming interface, many common mistakes that are encountered in imperative programming are completely avoided. One example is the "off by one" bug which, as you could imagine, would be a very important error to avoid when it comes to circuit design.

1.5 Features

1.5.1 Components

Components in LCSL can be thought of as a hybrid of classes and methods in Java. Components are defined like methods representing a specific function, but they can be instantiated and referenced like Java objects. Also like objects, components allow access to the parameters that are inputted and outputted to the function definition.

1.5.2 Recursion

Recursive programming has never been associated with circuit design… until now. With LCSL, you can recursively call components that make your design much easier to understand conceptually. For instance, if you created a bus using the same component several times (as we show in the tutorial), you can just recursively call that component instead of explicitly calling every single time. Another example is a component that has within its definition instances of itself forming a tree-like recursive structure. Recursion saves time and space and also is extremely simple. Anyone can look at the code and understand what the main component is supposed to do.

1.5.3 Overloading

Another great feature of LCSL is the concept of overloading. LCSL has several built-in functions that represent operations such as adding, subtracting, XORing, and ORing. However, all these operations are actually just built-in component definitions, basically making each operation symbol (such as + for adding) the name of the component. If you simply create a new component with the same name, your component overrides the built-in component in whatever scope you're working under. This can come in very handy when you want to slightly tweak a built-in operation for experimentation or simply don't need that built-in function and could use the operator for a completely different function that is more important to your project. Note, however, that names cannot be overloaded in the same environments where they already appear, or a duplication error occurs.

Chapter 2
Tutorial

2.1 Simple Example

Here is a simple example of LCSL code that will produce an XOR gate:

```
Xor <- comp +A +B -X
            X <- A '^' B
       end
Sys <- {Xor}
{VectorInput, "In1" 1 4, Sys.A}
{VectorInput, "In2" 2 4, Sys.B}
{VectorOutput, "Out" 3 Sys.X, _}
{Set, "BuildName" "OneBitXor", _}
{Set, "FileName" "xor.v", _}
{Set, "GenTestBench" true, _}
(Set, "BenchName" "testbench.v", _}
```

In this example, the first thing we do is define a component. A component can be though of as similar to a function, where parameters are inputted and a value or values are outputted. We define a component with the `comp` keyword. We input two variables, A and B, which are denoted with a `+` to signal they are "in ports". The sole output for this component is the variable `X` which is denoted with a `-` sign in front that signals an "out port".

Next, we assign to the out port variable `X` the expression `A '^' B`, where the operator `'^'` represents the built-in function XOR. The component definition is completed with the `end` keyword.

We then assign this component to another variable `Xor` using the operation `<-`. Doing this is similar to naming a function in Java or C, except that the name in this case is another variable. Components also act similar to Java objects. You can't directly access a component; instead you have to create an instance of it before you use it. This is what the next statement means: `Sys <- {Xor}`. We are defining an instance of `Xor` (which is a component) and assigning it to the `Sys` variable.

Now we explain how the in and out ports defined in the component are going to be translated into Verilog code which is our ultimate goal. Components in LCSL are generally translated into one Verilog module, and we must tell Verilog how

the ports should be defined for the module being built. This is what occurs in the next three lines. Since the in and out ports of this Xor gate are bit vectors, we use the built-in components VectorInput and VectorOutput to actually define what the ports' values are going to be. In the first line (`{VectorInput, "In1" 1 4, Sys.A}`) the number `1` represents the position in the port list. The number `4` in this case represents the size of the bit vector, and the string `"In1"` represents the name of the vector in the Verilog code. In this case, Sys.A, which is the first in port of the instance of an Xor component, will be used as the vector. The same is done in the next line, and then VectorOutput is used to define the out port in the Verilog code.

The last four lines use another built-in component called Set. In the first instance of Set, the user specifies what the main module will actually be called in Verilog. The next line also uses an instance of Set, but this time it is used to specify the name of the file that LCSL will output the Verilog code to. The third line of this block sets the flag GenTestBench to true which means that the user wants LCSL to generate a Verilog test bench file for simulation purposes. The last line specifies the file name of the test bench file.

2.2 Compilation/Simulation of Simple Example

Once the example has been saved into a file, we can begin the compilation process. LCSL takes in any ASCII text file, but for good organizational practice, we generally use the `.lcsl` suffix to denote LCSL code. In this case, we saved this code in the file `xor.lcsl`. To compile, we did the following (assuming the LCSL package is in the class path and we are in the same directory as the file):

```
$ java –jar LCSL.jar xor.lcsl
```

This should produce the file xor.v that has Verilog code.

In order to run the Verilog code, we use the testbench file we specified in the program. In our example, the testbench file was named `testbench.v`. To run Verilog we use the Icarus compiler:

```
$ icarus –o output testbench.v
```

The simulation should occur. Yes, we understand that for simulation you do need to know a little Verilog, but trust us, the code needed in test bench is very easy to understand and write, and LCSL saved you time on the really hard stuff which is the creation of the main module that you're simulating in the first place. Instead

of writing tedious Verilog code for it, a few lines in LCSL were all that was needed!

2.3 A more complex example… with recursion!

As mentioned in Chapter 1, LCSL can do recursion which shows how highly abstract this language is. In this example, we expand our 1-bit XOR unit created in the simple example to a 4-bit XOR unit using recursion (To tell you the truth, recursion is not necessary. In fact, if we didn't use recursion, we could actually write this in even less LCSL code, but for the sake of example, we'll continue):

```
Xor <- comp +A +B -X
              X <- A '^' B
           end

BusXor <- comp +A +B -X
                    if width A == 0
                       X <- <>
                    else
                      {Xor, 'lsb' A 'lsb' B, BitX}
                      {BusXor, 'msbs' A 'msbs' B, SubX}
                      X <- SubX '++' BitX
                    end
          end


Sys <- {BusXor}

{VectorInput, "In1" 1 4, Sys.A}
{VectorInput, "In2" 2 4, Sys.B}
{VectorOutput, "Out" 3 Sys.X, _}
{Set, "BuildName" "NBitXor"}
{Set, "FileName" "nbitxor.v"}
{Set, "GenTestBench" true, _}
(Set, "BenchName" "testbench.v", _}
```

In this example, we've redefined the XOR unit from the previous example in the first three lines. The next component we define which is BusXor makes things interesting. We have included an if statement which is necessary for any recursion example as a base case is needed to stop the recursion at some point. The component Xor is called for the least significant bit of the two "in port" vectors (which must be of equal size), and then BusXor is recursively called for the rest of the bits of the vector. The base case is when the rest of both vectors is of width 0 (meaning no bits), which in that case we assign a null vector to X denoted by <>. The result of the Xor component is sent to the variable BitX, and the result of the recursive line with BusXor is sent to the variable SubX. Then BitX is

concatenated to SubX and assigned to X, and we have finished this component definition. We then create an instance of this component and assign it to Sys and then use the same built-in components as in the simple example.

Notice how when using Xor and BusXor, we didn't actually have to create an instance of either when calling them. This is special syntactic sugar in LCSL. What is actually happening in both lines is that instances are in fact created and assigned to dummy variables. However, because the user wrote it this way, LCSL knows all the user cares about is the output variables, so we can basically discard the dummy variables once values are assigned to the output variables. This just makes life easier for the user since he doesn't have to explicitly create instances of a component that he's probably going to use only once and probably won't show up in the Verilog code anyway.

Chapter 3
Reference Manual

3.1 Lexical Conventions

3.1.1 Comments
Comments begin with `(*` and end with `*)`

3.1.2 Variables
Variables consist solely of alphanumeric characters and underscores. The first character must always be an uppercase letter (A-Z) and the last character can never be an underscore ("_").

3.1.3 Keywords
All keywords start with lowercase letters in order to avoid any overlap with variable names:

```
true            false           comp
if              else            end
```

3.1.4 Operators
A complete list of operators and their functions are included later in this chapter. The operators accepted by the lexer include:

```
.       `       !       ~       **      *       %       +       -       <<      >>
@       <       >       <=      >=      ++      &       ^       |       <-
```

3.1.5 Strings and Vectors
Strings use Java-like syntax with an ASCII character string surrounded by double quotes (ex. `"Hello World"`). Vectors are strings comprising of only 0s and 1s and are surrounded by < and > (ex. `<01001110>`). Both Strings and Vectors will be explained fully in the next section.

3.2 Types

LCLS includes common types as integer, Boolean, or string. Since it is a circuit design language, its more specific types are vectors, components, and systems. All types (less systems) can be passed in and out of components and can be operated on. For example, components can be passed into other components and instantiated in there. LCSL is not strongly typed in the sense that variables need

not be given a type in advance. In fact, they don't even have to be introduced; LCSL automatically introduces them when used. However, the types of the entries that variable resolve to are checked for consistency when various primitives are applied to them.

3.2.1 Integers
Integers are fundamental data types that can be expressed in decimal notation.

3.2.2 Booleans
Booleans are fundamental data types that can take on either a true or false value.

3.2.3 Vectors
Vectors are fundamental data types that are similar to bit-vectors in hardware design languages and consist of basic logical bit values (0 and 1). Vectors can be created by writing them directly between < and > and connecting this string to a variable.

3.2.4 Strings
Strings are the same syntax as in Java and C.

3.2.5 Components
Components are similar to functions in other languages that can pass into and return from other components. This is a primary feature of LCSL and is a characteristic of its functional programming nature. Components are declared using the comp keyword. Parameters passed to and from a component are known as ports. The ports of a component are preceded with either a '+' or '-' which mean an input or output port respectively. The inputs must be listed before any output port. The body of the component, which is a set of statements, follows the port definitions. Finally, the end keyword signals that the component definition is complete.

3.2.6 Systems
Systems represent an actual instance of a component. A variable becomes a Sys type when it is connected to an instance of a component with specific input and output port values given. Because systems are actually environments, they incur some constraints: they cannot be passed in and out of components, and must be available at the time hierarchical names are evaluated.

3.3 Type Operators

3.3.1 Integer Operators

The following is a table for operations of integers.

Operator	Function
`X * Y`	multiplication
`X / Y`	division
`X % Y`	modulo
`X + Y`	addition
`X - Y`	subtraction
`X < Y`	less than
`X > Y`	greater than
`X <= Y`	less than or equal
`X >= Y`	greater than or equal
`X == Y`	equal
`X != Y`	not equal

3.3.2 Boolean Operators

The following is a table for operations of Booleans.

Operator	Function
`! X`	logical NOT
`X & Y`	logical AND
`X | Y`	logical OR

3.3.3 Vector Operators

The following is a table for operations of vectors.

Operator	Function
`X ' [int]`	bit selection
`~ X`	bitwise negation
`'width' X`	vector width
`'msb' X`	most significant bit
`'msbs' X`	all bits except LSB
`'lsb' X`	less significant bit
`'lsbs' X`	all bits except MSB
`X '*' Y`	multiplication
`X '+' Y`	addition
`X '-' Y`	subtraction
`X '<<' [int]`	shift left
`X '>>' [int]`	shift right
`X '<' Y`	less than
`X '>' Y`	greater than
`X '<=' Y`	less than or equal
`X '>=' Y`	greater than or equal
`X '++' Y`	vector concatenation

`X '==' Y`	equal
`X '&' Y`	bitwise AND
`X '|' Y`	bitwise OR
`X '^' Y`	bitwise XOR
`X '!=' Y`	not equal

3.4 Expressions

3.4.1 Expression Syntax

Expressions in LSCL are based on the conceptual notion of source expressions and sink expressions. Source expressions produce values whereas sink expressions consume values.

3.4.2 Basic Expressions

3.4.2.1 Constants

Constants are the most fundamental of values and can be of either of the fundamental data type forms (integer, Boolean, Vector).

3.4.2.2 Variables

Variables are bounded to a specific value based on another expression. Unlike descriptive languages such as Java, a variable can only be bounded once and cannot change through the course of the program.

3.4.2.3 Instantiations

Instantiations, or systems, create instances of components with certain input and output port values based on other expressions. Instantiations as expressions are just a component name wrapped in braces. An example of an assignment where a variable is bound to a system:

```
X <- {Xor}
```

There are two ways of accessing the ports of this system:

```
X.a <- A
X@2 <- B
```

Ports can be accessed by name with the dot notation or by position in the port list by the @ notation.

Ports can also be assigned in the initial statement like this:

```
X <- {Xor, A B, X}
```

In this case, the name of the component is the first term, and the two in ports are bound to A and B as the second term. The third term is where we bound the out port to the vector X.

To access one or more ports later rather than specifying them in the instantiation system, simply place an underscore where an expression would have normally gone. For instance:

```
X <- {Xor, A _, X}
X <- {Xor, A B, _}
```

In the first case, the second in port is not specified, and in the second case, the out port is not specified. These ports can be specified later using the dot or @ notation mentioned above.

3.4.2.4 Operational expressions

Depending on the data type of the value of variables and sub-expressions used, there are several operations that create expressions. Logical AND (`&`) , OR (`|`), and XOR (`^`) can be used with several sub-expressions. The prefix unary operator `!` and infix binary operators like `+` are also ways to create expressions.

3.4.3 Source Expressions

As mentioned above, source expressions produce values. The following are valid source expressions:

- Variables
- Constants (integer, Boolean, or Vector)
- Connections
- Instantiations
- Conditionals
- Component definitions
- Operational expressions
- Expression groupings of any combination of above

3.4.3 Sink Expressions

As mentioned above, sink expressions consume values. Only variables may be used as outputs of source expressions.

3.5 Statements

3.5.1 Connections
Connections using `<-` , defined above as a type of expression, are also simple statements.

3.5.2 Conditionals
Conditionals as statements provide the same functionality as if they were basic expressions but also provide extra functionality through the else keyword. For instance:

```
if (<predicate>)
  (* other statements *)
else
  (* even more statements *)
end
```

3.5.2 Instantiations
Instantiations can be run on their own without being bounded to a sink variable. This is because the out ports of the system are sinks themselves, so there is at least one thing being outputted to a sink variable anyway. Here is an example of an instantiation as a statement:

```
{Xor, Input1 Input2 Input3, Output}
```

In this case, we ignore having to assign the actual system to a sink variable since in this case all we care about is the sink variable `Output`.

In case we want certain ports of the system not to be specified because they're unimportant, we can bypass certain ports with an underscore like this :

```
{Xor, Input1 _ Input3, Output}
```

Here the second in port is given no value.

There are a few built-in instantiation systems that we will mention now.

3.5.3 VectorInput and VectorOutput
This system is used to input a vector into port lists of modules created in Verilog. The syntax can be seen in the following example

```
{VectorInput, "Input1" 1 4, X}
```

The string "Input1" represents the name of the vector put in the port list in Verilog. The next parameter is 1 which represents the position within the port list

that's the vector is being assigned to. The next parameter is 4 which represents the width of the vector. Finally, the parameter that's in the output position of the system is the variable X which represents the bit vector in LCSL that will be used as the input vector.

Just like LCSL can define the input vector of the module written in Verilog, it can also define the output vector. The syntax can be seen in the following example:

```
{VectorOutput, "Output1" 3 A, _}
```

In this case the Verilog vector will be named "Output1" and is coming from the 3rd position of the port list. This vector will be assigned to the LCSL vector variable A.

3.5.4 Set

Another special and very important built-in instantiation system is Set. This allows the user to set necessary environmental variables such as the filename of the output Verilog file and the module name within that file. Here are all the tasks that can be accomplished with the Set system:

`{Set, "BuildName" [name], _}`	Sets the Verilog module name to [name] (default is UnnamedModule)
`{Set, "FileName" [name], _}`	Sets the Verilog file name to [name] (default is unnamedFile.v)
`{Set, "GenTestBench" [bool], _}`	Flag to generate testbench file, set to Boolean value [bool] (default is false)
`{Set, "BenchName" [name], _}`	Sets testbench file name to [name] (default name is unnamedBench.v)

Chapter 4
Project Plan

4.1 Team Responsibilities

Early on, we decided on four roles that would help us become specialized in a specific field and help make the project more efficient and organized. Here are all of our roles:

Bogdan Caprita	Complete Java back end, Verilog output
Sachin Nene	Documentation ANTLR Grammar/Parser/Walker
Julian Maller	ANTLR Grammar/Parser/Walker CVS Administrator
Chaue Shen	Tester, Confluence Tester

4.2 Programming Style Guide

Because all four of us specialized in different fields that were all essential to this project, we all added different pieces to the entire code of this language. The best way we found to keep things organized was using the source control system CVS to keep track of revisions. By using a strict approach to checking out and checking in source files, we avoided mix-ups and errors from different people editing the same piece of code.

Because the back end was completely done in Java, we utilized it's object-oriented philosophy by creating a good set of APIs. By creating well-defined classes and methods, the developers who worked on the ANTLR tree walker would not have to know the specifics on how the back end objects worked. Instead, they relied on the API and trusted the back end developer in implementing the appropriate methods.

The same could be said for the back end developer. He could be assured that the main object he was being fed from the tree walker would meet his specifications exactly. In other words, he did not have to worry about the syntactic sugar of the language; the walker developers transformed whatever form the LCSL source code was in into a well-structured unit that the back end could understand.

And as always, there was often an overlap in responsibilities when it came to both the front end parser and back end, which meant that more than one person, could be working on a specific piece of the source code. Explicit commenting that was time stamped along with the initials of the developer working on it insured that everyone was on the same page for that specific piece of code.

4.3 Project Timeline

Here is the project timeline that we followed to the best of our abilities:

September 23	Complete White Paper
October 16	Complete initial grammar, parser
October 23	Complete Language Reference Manual
November 10	Complete initial tree walker
December 7	Complete presentation preparation
December 10	Finalize front end (lexer, parser, walker)
December 12	Complete back end
December 15	Complete testing, error handling
December 18	Complete Report
December 19	Present to Professor Edwards

4.3 Software Project Environment

The entire back end was written in Java, and the lexer and parser was written in ANTLR. The Verilog code produced by the back end was tested on the Icarus Verilog compiler. All code was managed through CVS on a Linux machine.

4.3.1 Operating Systems

Because of the portability of the Java programming language, the current build of this language will work on any operating system that can run the Java Virtual Machine.

4.3.1 Java 1.4.2

We developed the entire back end in the Java programming language. We did not use any of the new features of 1.4.2, so our code should be compatible with older builds of the Java Runtime Environment and Standard Development Kit.

4.3.2 CVS

For source code control, we used Concurrent Version Systems, or CVS. We developed and managed all our code on Julian Maller's Linux machine which ran the CVS server as well.

4.3.3 Icarus

To test the Verilog code that we created from LCSL, we used the widely popular open source Icarus Verilog compiler. The main reason we chose this compiler was probably because it's free. The compiler was installed on Julian's Linux machine.

4.3.3 ANTLR

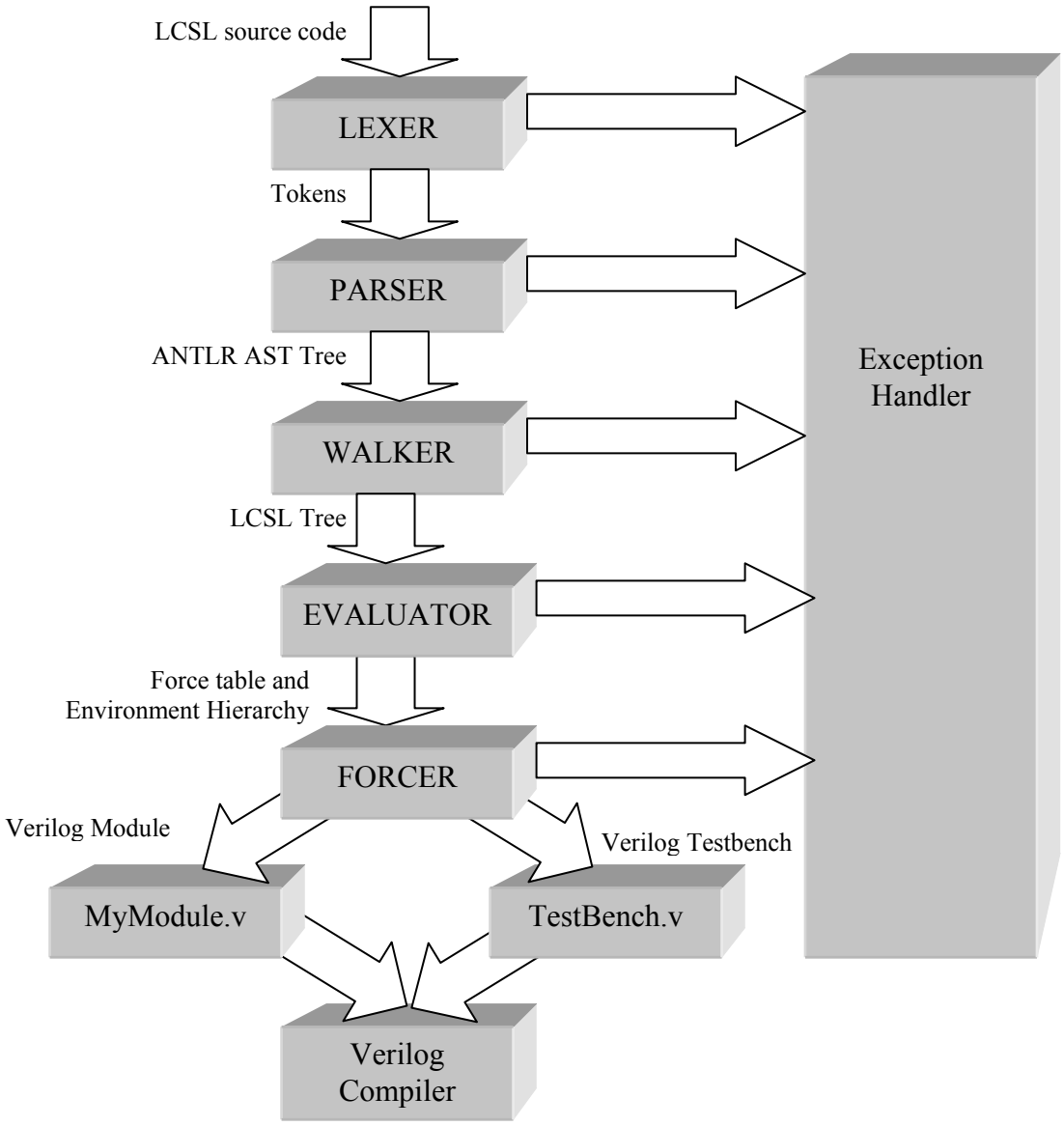
Our workload was drastically reduced by using ANTLR in the process of creating our grammar, parser, and walker. ANTLR syntax is simple to understand, and the language is powerful enough to correctly implement the abstract syntax tree we desired. Not only this, ANTLR was also allowed us to easily transform the AST to an object-based LCSL tree which was necessary as input for the back end.

4.4 Project Log

September 23	White paper written
Week of Oct 13	Initial grammar, parser for the language completed
October 23	Language Reference Manual written
Week of Nov 3	1st full revision of grammar and parser completed
Week of Nov 10	Initial tree walker completed
Week of Nov 17	1st full revision of walker completed Begin work on Java back end
Week of Nov 24	3rd full revision of lexer, parser completed continued work on 2nd revision of walker
Week of Dec 1	Begin work on presentation 2nd full revision of walker completed
Week of Dec 8	Finalized grammar, parser, walker completed Presentation given
Week of Dec 15	Back end completed Continued testing, error handling Final Report written
December 19	PROJECT COMPLETED

Chapter 5
Architecture Design

5.1 Block Diagram



5.2 Description of Architecture

5.2.1 The Functional Programming Model

LCSL is meant to leverage the functional programming paradigm in designing logical circuits. Although this approach is generally not as wide-spread as the sequential, imperative way of writing programs, we have found functional programming to be a very clean and intuitive abstraction of the connections upon which a logical circuit is built. In some aspects such as evaluation order, or variable binding, LCSL is more pure of a functional language than SCHEME or other dialects of Lisp. Not surprisingly, the architecture of the LCSL compiler finds its roots in the SCHEME interpreter described in the "Functional Programming Bible".[1]

In LCSL, the main vehicle of abstraction is the *function*. Since our language is geared towards hardware design, we have called our functions components. Like functions in SCHEME, components have inputs and a body that give the rules to bind the outputs to values. A key difference is that LCSL components can have multiple outputs, or none. This offers flexibility and power to the language, but complicates slightly the evaluation of LCSL programs, as we will discuss later. LCSL components are declared *anonymous*, and can optionally be bound to variables or passed as inputs to other components. In fact, in LCSL everything is a λ function/component, including language primitives such as Vector input/output declarations and operators, be they unary or binary infix (we have seen that `C <- A '+' B` is mere syntactic sugar for instantiating the primitive component named `'+'`).

5.2.2 β-reduction

The more interesting phase of compilation in LCSL is β-reduction. SCHEME allows for a simple evaluate/apply pattern as its execution cycle: evaluating entails looking up values for names, or resolving assignments, other language constructs such as decision blocks, and, potentially, function applications, whereas function applications involve evaluating the inputs (to obtain their values) and evaluating the body of the function sequentially. LCSL stays clear of this simple paradigm for two main reasons: the fan-out of components, and the extreme-lazy evaluation technique we found to be well-suited for our purposes.

---

[1] Harold Abelson, Jay Sussman, and Julie Sussman. Structure and Interpretation of Computer Programs, 2nd ed, MIT Press, 1996

First, components are not restricted to a single output. This is a natural property of the functional unit that we are abstracting (sub-circuit as a black box with wires going in and out), but involves separate steps to bind each of the outputs. Second, circuit design is not a streamlined, stepwise process: components are designed and wires are connected in no predefined order. Asking the designer to specify and assign values to all of the variables used seems an artificial constraint that can be an unnecessary burden to the electrical engineer unaccustomed to mainstream imperative programming. Therefore, while in SCHEME applying a function is synonymous to setting up a new environment, evaluating its arguments and body and returning a value, we have separated the instantiation of a component from the binding of its ports and from the actual evaluation and value-forcing.

When a component is instantiated into a *system*, the only action is to resolve the component name being instantiated and to set up the environment frame associated with this component application. The first step therefore requires components to be defined in the body before they are instantiated. As a corollary to the second step, using hierarchical names requires that the systems referenced be instantiated beforehand (since the hierarchical names need to resolve to a name and an environment of evaluation). These are the only two constraints placed on the order of statements in a LCSL program. Thanks to the *Thunk* design, connections can be made and variables used before any values are available. In particular, inputs to components are forced for their values only when there is no other way out. In fact, we go a step beyond normative evaluation: not only inputs to components need not be all defined at the time the component is instantiated into a system, but some systems and assignments will never get evaluated or forced for values at all. Forcing occurs only on demand of another forcing, and the chain of forcing originates at names used in VectorInput and VectorOutput primitives. The reason for this approach is that the ultimate Verilog module's interface to the outside Verilog world is its input and output ports. As long as the outputs get their prescribed values, other, secondary computation is ignored if not related to the branches producing the outputs.

5.2.3 Variable binding and name lookup

In LCSL, all names are resolved through environments. The frames (symbol tables) or environments create a hierarchy. There is a global environment in which the main body of a program is evaluated, as well as where the built-in primitive components are linked to their names. Each component instantiation has its own environment, whose parent is the environment where the component was defined. Thus, systems are little more than environments where the body of the corresponding component is evaluated. LCSL uses *deep binding*. LCSL variables

are simply entries in symbol tables, and table entries can be roughly divided into two categories: vectors and scalars. A vector is similar to a Verilog wire, and any operations involving vectors (either in the form of wires or literal 0/1 bit vectors) are either optimized away by the compiler or converted to corresponding Verilog code that would perform de desired calculations. Scalars (integers, Booleans, or character strings) are forced to actual literal values by the compiler and are used exclusively for the purpose of control flow. We can think of vectors as the Verilog objects that are laid out in the output module. They cannot be used in control flow such as decision blocks, since values are not assigned to wires until the Verilog compiler gets to act on the output module.
LCSL strictly respects the unique binding rule: names can be assigned to at most once. This helps the conceptual design of circuits, where wires, once laid out, are there to stay.

5.2.4 Evaluation and forcing

The evaluation step passes through the LCSL Expression tree and does a minimal amount of work, the most important of which are setting up environments for systems and creating *thunks* for names. Thunks were coined by the Algol-60 team to represent call-by-name and means "thought about." It is essentially a name/environment pair that specifies the environment where a name needs to be forced in order to retrieve an actual final value. Thunks are set up for most variables. The evaluation step also remembers in the *force table* which names absolutely *need* to be forced (i.e., resolved to values). These are generally inputs to VectorInput and VectorOutput, as discussed above.
The forcing phase than initiates recursive forcing of thunks that call up a chain of component body evaluations, variable lookups, primitive applications, or other thunk forcing. Some evaluation braches may never be traversed, If they don't contribute to the operation of the resulting Verilog module. This justifies the extreme-lazy evaluation strategy employed. We illustrate this with the following example:

```
A <- B
B <- A
C <- A '+' B
{VectorOutput, "Out" 1 3, D}
```

If the above code contains something like `C <- D`, then the compiler will try to resolve A and B and output an error message when it detects a circular dependency or variables. If, on the other hand, `C <- E` for some E, the code involving A and B will not be evaluated and the program will output the Verilog code with the correct functionality intended by the designer.

5.3 Authors of Components

The entire back end system which is encapsulated in what we call the evaluator and forcer was designed and implemented by Bogdan Caprita. The grammar was worked on by all four members of the group, and the parser and walker were designed and implemented by Julian Maller and Sachin Nene. Exception handling in the back end was done by Bogdan while exception handling on the front end was automatically done for us by ANTLR.

Chapter 6
Testing Plan

6.1 Early Testing

The LCSL syntax is based on the syntax of the Confluence hardware design language which is another HDL that uses a functional programming interface for abstract design.

As the parser and lexer were first being built, early Confluence examples were written to ensure that they were working. These early Confluence files were not syntactically correct but rather were written to test specific aspects of the language. For example, the following is an excerpt from one such file:

```
A <- Aavkasdf
A <- 4584
A <- Foo
$ <- true (* user defined var can be any char so long as it's not
a keyword*)
! <- 34
Q <- true
Q <- !false
Q <- maybe
Z <- A + 1
Z <- A || B
Z <- ! Sys.port
Z <- Sys.Port
Sys.port <- Sys@34
Sys.Port <- Sys@port
Z <- Sys@Pavd (*Pavd should be required to be an integer I think
*)
Z <- prim Target +A +B -C end
Z <- "string"
A <- "'string'"
A <- "$%@#$^%$+*#%^@|\/?<>,.adfl;a;vadsf"
Z <-
  comp +A +B -X
     X <- A + B
  end
```

Other files were used to test various components and declarations. The contents of the files slowly grew as the grammar became more robust.

6.2 Our compiler works!

After we got our compiler to work, we wrote several programs in syntactically correct LCSL. Eventually, these files would be used for regression testing. Our first examples were a simple XOR gate and an XOR gate implemented through recursion. While logically simple, our first LCSL programs were very momentous and verified that our compiler actually outputted Verilog code. At this point we had to verify our Verilog code manually since we had not learned how to use Icarus or how to write a test bench for Verilog. Despite this, several more test files were written in LCSL and slowly a suite of files was put together as our "test suite." The complete list of test files is included in Appendix A, Section A.4.

6.3 Running Verilog Code

In order to test our Verilog we had to install Icarus and write test benches for each of our LCSL files. In order to streamline the process we added functionality to the compiler that would automatically generate a test bench for the outputted Verilog. To do this, insert the following two lines of code to your LCSL files: To actually run and test the Verilog enter the following two command prompt commands after compiling the LCSL file.

```
$ iverilog –o <output file> <test bench file>
$ ./<output file>
```

This will output all possible combinations of inputs to the circuit (inputs are all vectors using the VectorInput primitive) and the outputs that the particular sequence of inputs generates.

6.4 Regression Testing

After having a suite of test cases and the ability to generate the truth tables for the Verilog our compiler outputted for the test cases we could then write a script to use for sequential testing. This script simply compiles the test suite and runs the outputted Verilog. Finally the resulting output from the testbench is compared to previously verified output stored in specific files.

In doing so we were able to make changes to the compiler and automatically test to see if our new changes had caused any problems with code that was working previously. As our compiler grew more robust more test cases were created.

Sequential testing was also very useful in testing alternative syntax because altering a LCSL file without altering the logic would not change the resulting

truth table. An altered LCSL file with unchanged logic could be run through the script and the verified output would still be expected to hold. The regression testing script written in Perl is included in Appendix A, Section A.4.

Chapter 7
Lessons Learned

7.1 Each Member's Lessons Learned

Sachin Nene- Working on LCSL this semester was quite an experience. I had never worked in a team programming project quite like this before. Organization became crucial, and we had to become really disciplined when it came to good source code control. I also realized early on that compromises had to be made for the different ideas and directions each teammate had in mind. Scheduling was also an important aspect of this project; being able to meet not only to discuss the progress of the project but also be able to code together so that we could help one another.

Chaue Shen- LCSL taught me a lot about working in group environments. Although we did divide the job amongst ourselves there ended up to be plenty of overlap. This forced me to get comfortable with other people's coding styles, which helped me test in the end. Picking a group leader was also very important to the project. Without a leader, or when our leader wasn't adamant about his ideas, everyone seemed to put his two cents in which ultimately caused hour long discussions about the smallest details. Lastly, I found it very difficult as tester to keep up to date on how the different aspects of the project were going. In order to test effectively I had to keep an eye on each part and imagine how things would fit together in the end.

Bogdan Caprita- Working on a group project over an extended period of time, I have come to value the advantages of team work. Knowing that I could rely on my group members to provide the backend I was writing with the precise input we had agreed on and that the unavoidable bugs in my code would be eventually caught by the tester boosted my confidence in the success of our project. Knowing that a part of the system depends on me motivated me to do my best to deliver the correct behavior for the LCSL programs, even if some aspects seemed very challenging at first. When I got carried away to propose unrealistic design features given our resource constraints, my team members promptly sanctioned unrealizable goals and instead proposed better alternatives.

I learned that a good project needs good humor to get it moving. Bad humor can also help, though, especially in coping with the initial difficulties of each step of the project. We had plenty of both.

Julian Maller- During the course of this project I learned several important lessons about working in a group. The first lesson was communication. Even when the project is broken up so that different people work on different parts, they all need to interact at some point, and if I decide that I'm going to expect a certain input and not bother to tell the other programmers, we're going to run into problems. Part of this means making sure to schedule regular meetings. The other major lesson was to make an overall plan for executing the project, and stick to it. This does not mean you cannot change the plan, but it is important to always have a goal and a plan for how to reach that goal.

Appendix A
Source Code

A.1 Front End

A.1.1 Lexer

```
header {
    package LCSL;
    import java.util.*;
}

/********************
grammar.g

@author Sachin Nene (srn28@columbia.edu) and Julian Maller
(jules@columbia.edu)

This file houses the lexer, parser, and walker for LCSL

*****************/



class LCSLParser extends Parser;

options {
    k = 3;
    buildAST = true;
    exportVocab = LCSLAntlr;
}

tokens {
    PROGRAM;
    COMPONENT_INPUTS;
    COMPONENT_OUTPUTS;
    IN_PORTS;
    OUT_PORTS;
    BODY;
    INST_SYS;
    INST_SYS_PORTS;
}


statements
    :   (statement statements
        | //nothing
```

```
        )
;

statement
    :   ( ifelse
        | (connection)=>connection
        | instantiation_system
        )
;

term_literal
    :   (
         STRING
        | INTEGER
        | "true"
        | "false"
        | BITVECTOR
         )
;


expression_source
    :   ( term_literal
        | identifier
        | NAME VBSEL^ INTEGER
        | component_anonymous
        | (instantiation_system)=> instantiation_system
        | LPAREN! expression_compound RPAREN!
        )
;

expression_sink
    :   ( identifier  )

;

identifier:
        (NAME | NAME AT^ (NAME | INTEGER) | NAME PERIOD^ NAME)
;



//The following is the precedence for unary and binary operators

expression_compound
    : (equal_expr ((AND^ | XOR^ | OR^ | VBAND^ | VBOR^ | VBXOR^)
equal_expr)*)
;

equal_expr
```

```
    : plusplus_expr ((DEQ^ | NEQ^ | VEQ^ | VNEQ^) plusplus_expr)*

;

plusplus_expr
    : compare_expr ((VCAT) compare_expr)*
;

compare_expr
    :   shift_expr ((LT^ | GT^ | LEQ^ | GEQ^ | VULT^ | VUGT^ |
VULEQ^ | VUGEQ^) shift_expr)*
;

shift_expr
    :   add_expr ((LSHIFT^ | RSHIFT^ | VLSHIFT^ | VRSHIFT^)
add_expr)*
;

add_expr
    :   mult_expr ((PLUS^ | MINUS^ | VADD^ | VSUB^) mult_expr)*
;

mult_expr
    :   power_expr ((MULT^ | FSLSH^ | MOD^ | VUMUL^) power_expr)*
;

power_expr
    :   unary_expr (POWER^ unary_expr)*
;

unary_expr
    :    (VWIDTH^ | VMSB^ | VMSBS^ | VLSB^ | VLSBS^ | NOT^ |
VBNEG^ | TILDE^)? expression_source
;

ifelse
    :  ( "if"^  expression_compound ("then")?! body ifelse_else )
;

body
    : statements  {#body = #([BODY, "body"], body);} ;

program
    :   body EOF!
;

ifelse_else
    : (  "end"!
       | "else"! body "end"!
      )
```

```
;

//named anonymous because originally LCSL was planning to have
named components
component_anonymous
    :  ( "comp"^ component_inputs component_outputs body "end" )
;

component_inputs
    :   (PLUS! identifier)*
        {#component_inputs = #([COMPONENT_INPUTS, "comp_inputs"],
component_inputs);}
;

component_outputs
    :   (MINUS! identifier)*
        {#component_outputs =
#([COMPONENT_OUTPUTS,"comp_outputs"], component_outputs);}
;

//instantiation of a component
instantiation_system
    :  ( LCURLY! NAME RCURLY! {#instantiation_system =
#([INST_SYS, "inst_sys"], instantiation_system);})
        | (LCURLY! NAME COMMA! in_ports COMMA! out_ports RCURLY!
            {#instantiation_system = #([INST_SYS_PORTS,
"inst_sys_ports"], instantiation_system);})
;

in_ports
      : ( expression_compound | UNDERSCORE )*
        {#in_ports = #([IN_PORTS,"in_ports"], in_ports);}
;

out_ports
      : (expression_sink | UNDERSCORE)*
        {#out_ports = #([OUT_PORTS,"out_ports"], out_ports);}
;

connection
    :   expression_sink ARROW^ expression_compound
;

local_variable_inst
    :   COLON^  NAME;


null_terminal
    :   "null"
;
```

A.1.2 Parser

```
class LCSLLexer extends Lexer;

options{
    k=5;
    charVocabulary = '\3'..'\377';
    testLiterals = false;
    exportVocab = LCSLAntlr;
}

protected
LETTER
  : ('a'..'z') | ('A'..'Z');

protected
DIGIT
  : ('0'..'9')
;
protected
DIGITS
    :  (DIGIT)+
;

protected
EXPONENT
  : 'e' ('+' | '-')? (DIGIT)+
  ;

Whitespace
  : (' ' | '\t' | '\f')+
    { $setType(Token.SKIP); }
  ;


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


COMMENT
   :  ( "(*" (
                options {greedy = false;} :
                (NEWLINE)
                | ~( '\n' | '\r' )
             )*
        "*)"
```

```
        ) {$setType(Token.SKIP); }
  ;


//name can be a variable name or an operator preceded by the
character `
NAME options { testLiterals = true; }
  : ( 'A'..'Z' ) (( LETTER | DIGIT | UNDERSCORE )* ( LETTER |
DIGIT))?
     | ('`' (OPERATION))
;

KEYWORD options {testLiterals=true;}
    :   ('a' ..'z')+
;

INTEGER
    :   REGULAR_INTEGER

;

protected
REGULAR_INTEGER
    :   (DIGIT)+
;


BITVECTOR
    : ('<'! ('0' | '1')* '>'!)
;


STRING
    :   '"'! ( '"'! '"' | ~('"')  )* ( '"'!)
;

//all built-in operations
protected
OPERATION
    :   MINUS | POWER | MULT | FSLSH | MOD | PLUS | LSHIFT |
RSHIFT
        | LT | GT | LEQ | GEQ | DEQ | NEQ | AND | XOR | OR | NOT

        | VBSEL | VBNEG | VWIDTH | VMSB | VMSBS | VLSB | VLSBS |
VUMUL | VADD
        | VSUB | VLSHIFT | VRSHIFT | VULT | VUGT | VULEQ | VUGEQ
        | VCAT | VEQ | VNEQ | VBAND | VBXOR | VBOR
;
```

```
MINUS  : '-';
POWER  : "**";
MULT   : '*';
FSLSH  : '/';
MOD    : '%';
PLUS   : '+';
LSHIFT : "<<";
RSHIFT : ">>";
LT     : '<';
GT     : '>';
LEQ    : "<=";
GEQ    : ">=";
DEQ    : "==";
NEQ    : "!=";
AND    : '&';
XOR    : '^';
OR     : '|';
NOT    : '!';
TILDE  : '~';
PERIOD : '.';
QUES   : '?';
COLON  : ':';
AT     : '@';
ARROW  : "<-";
LBRKT  : '[';
RBRKT  : ']';
LPAREN : '(';
RPAREN : ')';
LCURLY : '{';
RCURLY : '}';
DOLLAR : '$';
COMMA  : ',';
UNDERSCORE : '_';


VBSEL  : '\'';
VBNEG  : "'~'";
VWIDTH : "'width'";
VMSB   : "'msb'";
VMSBS  : "'msbs'";
VLSB   : "'lsb'";
VLSBS  : "'lsbs'";
VUMUL  : "'*'";
VADD   : "'+'";
VSUB   : "'-'";
VLSHIFT: "'<<'";
VRSHIFT: "'>>'";
VULT   : "'<'";
VUGT   : "'>'";
VULEQ  : "'<='";
```

```
VUGEQ  : "'>='";
VCAT   : "'++'";
VEQ    : "'=='";
VNEQ   : "'!='";
VBAND  : "'&'";
VBXOR  : "'^'";
VBOR   : "'|'";
```

A.1.3 Walker

```
/*
This is the ANTLR AST tree parser that converts the AST to a LCSL
object tree
*/

class LCSLWalker extends TreeParser;
options {
    buildAST = true;
    importVocab = LCSLAntlr;
}

{ NameGenerator nameGen = new NameGenerator();  }

//main function called by the driver
body returns [Body b]
    { Body x; Vector v = new Vector(); b= new Body();  }
    : #(BODY (x=statement[b] {if (x != null) {b.addAll(x);}})*)
;

statement [Body theBody] returns[Body s]
    { s = null; Expression fromInst;}
    :   (s=connection)
     |  (s=ifstatement)
     |  (fromInst=inst_system_with_ports[theBody, null] {if
(fromInst != null) {s = (Body) fromInst;}})
;

ifstatement returns[Body b]
    {
        b = new Body();
        IfStatement i;
        Expression e;
        Body b1, b2;
    }
    :   (#("if" e=expr[b, null] b1=body b2=body) { i = new
IfStatement(e, b1, b2); b.add(i);})
;

expr_sink returns [HierarchicalVariable r]
    { r = null;}
```

```
    :   (r=identifier)
;

expr [Body parentBody, HierarchicalVariable sink] returns
[Expression r]
   {r = null;
    HierarchicalVariable hv = new HierarchicalVariable();}
   :  (#(INST_SYS hv=identifier) { Sys s = new Sys(hv); r = s; }
)
    | (str:STRING                { Str s = new
Str(str.getText()); r = s; } )
    | (i:INTEGER                 { Number n = new
Number(Integer.parseInt(i.getText())); r = n; } )
    | ("true"                    { r = new Bool(true);})
    | ("false"                   { r = new Bool(false);})
    | (k:BITVECTOR               { r = new BitVector(k.getText(),
k.getText().length());})
    | (r=comp)
    | (r=identifier)
    | (r=infix_expr[parentBody])
    | (r=inst_system_with_ports[parentBody, sink])
;


//this rule is for both infix and unary operators
infix_expr [Body parentBody]  returns [Expression r]
    { Expression b=null, c=null; r= null;  String op = "";  }
    : (
       (#(MINUS b=expr[parentBody, null] c=expr[parentBody, null]
) {op = "-";  } ) |
       (#(POWER  b=expr[parentBody, null]
c=expr[parentBody,null]    ) {op = "**"; } ) |
       (#(MULT b=expr[parentBody, null] c=expr[parentBody, null]
) {op = "*";  } ) |
       (#(FSLSH b=expr[parentBody, null] c=expr[parentBody, null]
) {op = "/";  } ) |
       (#(MOD b=expr[parentBody, null] c=expr[parentBody, null]
) {op = "%";  } ) |
       (#(PLUS b=expr[parentBody, null] c=expr[parentBody, null]
) {op = "+";  } ) |
       (#(LSHIFT b=expr[parentBody, null] c=expr[parentBody,
null]    ) {op = "<<"; } ) |
       (#(RSHIFT b=expr[parentBody, null] c=expr[parentBody,
null]    ) {op = ">>"; } ) |
       (#(LT  b=expr[parentBody, null] c=expr[parentBody, null]
) {op = "<";  } ) |
       (#(GT b=expr[parentBody, null] c=expr[parentBody, null]
) {op = ">";  } ) |
       (#(LEQ b=expr[parentBody, null] c=expr[parentBody, null]
) {op = "<="; } ) |
```

```
        (#(GEQ b=expr[parentBody, null] c=expr[parentBody, null]
) {op = ">="; } ) |
        (#(DEQ b=expr[parentBody, null] c=expr[parentBody, null]
) {op = "=="; } ) |
        (#(NEQ b=expr[parentBody, null] c=expr[parentBody, null]
) {op = "!="; } ) |
        (#(AND b=expr[parentBody, null] c=expr[parentBody, null]
) {op = "&"; } ) |
        (#(XOR b=expr[parentBody, null] c=expr[parentBody, null]
) {op = "^"; } ) |
        (#(OR  b=expr[parentBody, null] c=expr[parentBody, null]
) {op = "|"; } ) |
        (#(NOT b=expr[parentBody, null]
) {op = "!"; } ) |
        (#(TILDE  b=expr[parentBody, null] c=expr[parentBody,
null]    ) {op = "~"; } ) |
        (#(VBSEL  b=expr[parentBody, null] c=expr[parentBody,
null]    ) {op = "'"; } ) |
        (#(VBNEG  b=expr[parentBody, null]
) {op = "'~'"; } ) |
        (#(VWIDTH b=expr[parentBody, null]
) {op = "'width'"; } ) |
        (#(VMSB   b=expr[parentBody, null]
) {op = "'msb'";   } ) |
        (#(VMSBS  b=expr[parentBody, null]
) {op = "'msbs'";  } ) |
        (#(VLSB   b=expr[parentBody, null]
) {op = "'lsb'";   } ) |
        (#(VLSBS  b=expr[parentBody, null]
) {op = "'lsbs'";  } ) |
        (#(VUMUL b=expr[parentBody, null] c=expr[parentBody, null]
) {op = "'*'";   } ) |
        (#(VADD b=expr[parentBody, null] c=expr[parentBody, null]
) {op = "'+'";   } ) |
        (#(VSUB b=expr[parentBody, null] c=expr[parentBody, null]
) {op = "'-'";   } ) |
        (#(VLSHIFT b=expr[parentBody, null] c=expr[parentBody,
null]   ) {op = "'<<'";  } ) |
        (#(VRSHIFT b=expr[parentBody, null] c=expr[parentBody,
null]   ) {op = "'>>'";  } ) |
        (#(VULT b=expr[parentBody, null] c=expr[parentBody, null]
) {op = "'<'";   } ) |
        (#(VUGT  b=expr[parentBody, null] c=expr[parentBody, null]
) {op = "'>'";   } ) |
        (#(VULEQ b=expr[parentBody, null] c=expr[parentBody, null]
) {op = "'<='";  } ) |
        (#(VUGEQ b=expr[parentBody, null] c=expr[parentBody, null]
) {op = "'>='";  } ) |
        (#(VCAT b=expr[parentBody, null] c=expr[parentBody, null]
) {op = "'++'";  } ) |
```

```
       (#(VEQ b=expr[parentBody, null] c=expr[parentBody, null]
) {op = "'=='";  } ) |
       (#(VNEQ b=expr[parentBody, null] c=expr[parentBody, null]
) {op = "'!='";  } ) |
       (#(VBAND b=expr[parentBody, null] c=expr[parentBody, null]
) {op = "'&'";  } ) |
       (#(VBXOR b=expr[parentBody, null] c=expr[parentBody, null]
) {op = "'^'";  } ) |
       (#(VBOR b=expr[parentBody, null] c=expr[parentBody, null]
)  {op = "'|'";  } )
     )
       {
           Sys s = new Sys(new HierarchicalVariable("`" + op));
           Variable[] inputA = new Variable[2];
           Variable[] inputB = new Variable[2];

           String dummyVarName = "dummy" +
nameGen.getNextNumber();
           HierarchicalVariable sink = new
HierarchicalVariable(dummyVarName);

           parentBody.add(new Assignment(sink, s));

           HierarchicalVariable port1 = new
HierarchicalVariable(sink,  "1");
           HierarchicalVariable port2 = new
HierarchicalVariable(sink , "2");
           HierarchicalVariable port3 = new
HierarchicalVariable(sink , "3");
           Assignment assgn1 = new Assignment(port1, b);

           parentBody.add(assgn1);
           if (c != null ) {
               Assignment assgn2 = new Assignment(port2, c);
               parentBody.add(assgn2);
               r = port3;
           }
           else r = port2;

}
;

//identifier can be a variable name or a call to the ports of a
system (with @ or .)
identifier returns [HierarchicalVariable v]
  { Variable[] a; v = new HierarchicalVariable();}
    :  (#(AT
          (c:NAME {a= new Variable[2]; a[0] = new
Variable(c.getText());})
```

```
          (d:NAME {a[1]=new Variable(d.getText());} | h:INTEGER
{a[1]=new Variable(h.getText());})
         )
            {v = new HierarchicalVariable(a);}
        )

      | (#(PERIOD e:NAME f:NAME)
            {a= new Variable[2]; a[0] = new
Variable(e.getText());
             a[1]=new Variable(f.getText()); v = new
HierarchicalVariable(a);}
        )

      | ((g:NAME) {v=new HierarchicalVariable(g.getText());} )

;

inst_system_with_ports [Body parentBody, HierarchicalVariable
sink] returns [Expression r]

   {r = null; HierarchicalVariable inst_name;
    Vector inports, outports;
    Sys s;
    HierarchicalVariable dummyVar;
    Assignment assgn;
   }
        : #(INST_SYS_PORTS inst_name=identifier
inports=system_in_ports[parentBody]
outports=system_out_ports[parentBody])
            {
                Variable[] v;
                int portCount = 1;
                s = new Sys(inst_name);

                if (sink != null) {
                    parentBody.add(new Assignment(sink, s));
                    r = null;
                }
                else {
                    String varName = "dummy" +
nameGen.getNextNumber();
                    sink = new HierarchicalVariable(varName);
                    assgn = new Assignment(sink, s);
                    parentBody.add(assgn);
                    r = null;
                }

               //IN PORTS
                for (int n = 0; n < inports.size(); n++) {
                    if (inports.elementAt(n) == null) {
```

```
                        portCount++;
                        continue;
                    }
                    Expression e = (Expression)
inports.elementAt(n);
                    String portNumString = (new
Integer(portCount)).toString();
                    assgn = new Assignment(new
HierarchicalVariable(sink, portNumString), e);
                    parentBody.add(assgn);
                    portCount++;
                }

                //OUT PORTS
                for (int n = 0; n < outports.size(); n++) {
                    if (outports.elementAt(n) == null) {
                        portCount++;
                        continue;
                    }
                    HierarchicalVariable hv =
(HierarchicalVariable) outports.elementAt(n);
                    String portNumString = (new
Integer(portCount)).toString();
                    assgn = new Assignment(hv, new
HierarchicalVariable(sink, portNumString));
                    parentBody.add(assgn);
                    portCount++;
                }
            }
;

system_in_ports[Body parentBody] returns[Vector e]
    { e = new Vector(); Expression x;}
    :   #(IN_PORTS ((x=expr[parentBody, null] { e.add(x); }) |
UNDERSCORE {e.add(null);} )+ )
;

system_out_ports[Body parentBody] returns[Vector e]
    { e = new Vector(); HierarchicalVariable x;}
    :   #(OUT_PORTS ((x=expr_sink {e.add(x);}) | UNDERSCORE
{e.add(null);})+)
;

//also known as assignment
connection returns [Body b]
     {  b = new Body();
        HierarchicalVariable hv;
        Assignment a;
        Expression e;
     }
```

```
    : (#(ARROW hv=identifier e=expr[b, hv] )
        { if (e != null) {
                    a=new Assignment(hv,e); b.add(a);}
        }

      )
;

comp returns [Component r]
    {Vector a,b; Body c = new Body(); r = new Component();}
    : #("comp" a=comp_inputs b=comp_outputs c=body)
    { r = new Component(a,b,c); }
;


comp_inputs returns [Vector r]
     {r = new Vector();}
    :  #(COMPONENT_INPUTS (n:NAME {r.add(new
Variable(n.getText()));} )+)

;

comp_outputs returns [ Vector r]
     {r = new Vector();}
    : #(COMPONENT_OUTPUTS (n:NAME {r.add(new
Variable(n.getText()));} )+ )
;
```

A.2 Back End

A.2.1 Admin.java

```
package LCSL;

import java.util.*;
import java.util.regex.*;
import java.io.*;

/**
 * The main driver for the backend
 *
 * creates the global environment, adding all the language
primitives
 *   to it
 * manages the force table and the forcing of this table
 * handles the creation of the Verilog module output file
 * handles the creation of the Verilog test bench file
 *
 * @author Bogdan Caprita - bc2008@columbia.edu
 */
```

```
public class Admin
{
    private static Vector rememberTable;
    private static String fileName = null;
    private static String benchName = null;
    private static String moduleName = null;
    private static Vector portDecl, varDecl;
    private static Vector instructions;
    public static boolean debug = false;
    public static boolean genTestBench = false;


    /**
     * Inner class to handle port and variable declarations for
     * the resulting Verilog
     *
     */
    private static class Declar implements Comparable
    {
      private String name;
      private int position, width;
      private int type;
      private static Pattern pattern =
          Pattern.compile("[a-zA-Z_][a-zA-Z_0-9\u0024]*");

      public int compareTo(Object o)
      {
          int p = ((Declar)o).position;

          if (position > p){
            return 1;
          }else if (position < p){
            return -1;
          }else
            return 0;
      }

      public Declar(String s, int w, int p, int t)
      {
          name = s;
          position = p;
          width = w;
          if (width < 1){
            System.err.println(s+" has width "+w);
          }
          type = t;
      }
```

```
      public int getInputWidth()
      {
          if (type == 0){
            return width;
          } else {
            return -1;
          }
      }

      public static void validate(String s) throws
BadVerilogException
      {
          Matcher m = pattern.matcher(s);

          if (! (m.matches())){
            System.err.println(s+" is not an acceptable Verilog
identifier");
            throw new BadVerilogException();
          }
      }

      public static void validate(String s, Vector v) throws
DuplicationException, BadVerilogException
      {
          validate(s);
          for (int i = 0; i<v.size(); i++){
            Declar d = (Declar)v.get(i);
            if (d.name.compareTo(s) == 0){
                System.err.println(s+" is a duplicate
identifier");
                throw new DuplicationException();
            }
          }
      }

      public String toString()
      {
          return name;
      }

      public int getPos()
      {
          return position;
      }

      public String declare()
      {
          String size = "";
          if (width > 1){
            size += "["+(width-1)+":0] ";
```

```
        }
        switch (type){
        case 0: return "input "+size+name+";";
        case 1: return "output "+size+name+";";
        case 2: return "wire "+size+name+";";
        }
        return null;
      }

      public String declareBench()
      {
        String size = "";
        if (width > 1){
          size += "["+(width-1)+":0] ";
        }
        switch (type){
        case 0: return "reg "+size+name+";";
        case 1: return "wire "+size+name+";";
        }
        return null;
      }
    }

    public static void addInPortDecl(String s, int w, int pos)
throws DuplicationException, BadVerilogException
    {
      Declar.validate(s, portDecl);
      portDecl.add(new Declar(s, w, pos, 0));
    }

    public static void addOutPortDecl(String s, int w, int pos)
throws DuplicationException, BadVerilogException
    {
      Declar.validate(s, portDecl);
      portDecl.add(new Declar(s, w, pos, 1));
    }

    public static void addVariableDecl(String s, int w) throws
DuplicationException, BadVerilogException
    {
      Declar.validate(s, varDecl);
      varDecl.add(new Declar(s, w, 0, 2));
    }

    public static void addInstruction(String s)
    {
      instructions.add(s);
    }

    public static void remember(Vector thunks)
```

```
    {
      rememberTable.addAll(thunks);
    }

    public static Environment createGlobalEnv()
    {
      Environment global = new Environment("global");
      /* add entries for the primitives */
      try{
          PrimitiveSet pset = new PrimitiveSet();
          global.setValue(pset.getName(), pset);

          PrimitiveBinOp peq = new PrimitiveBinOp("`==");
          global.setValue(peq.getName(), peq);

          PrimitiveBinOp pneq = new PrimitiveBinOp("`!=");
          global.setValue(pneq.getName(), pneq);

          PrimitiveBinOp plt = new PrimitiveBinOp("`<");
          global.setValue(plt.getName(), plt);

          PrimitiveBinOp pgt = new PrimitiveBinOp("`>");
          global.setValue(pgt.getName(), pgt);

          PrimitiveBinOp pleq = new PrimitiveBinOp("`<=");
          global.setValue(pleq.getName(), pleq);

          PrimitiveBinOp pgeq = new PrimitiveBinOp("`>=");
          global.setValue(pgeq.getName(), pgeq);

          PrimitiveBinOp pplus = new PrimitiveBinOp("`+");
          global.setValue(pplus.getName(), pplus);

          PrimitiveBinOp pminus = new PrimitiveBinOp("`-");
          global.setValue(pminus.getName(), pminus);

          PrimitiveBinOp pdiv = new PrimitiveBinOp("`/");
          global.setValue(pdiv.getName(), pdiv);

          PrimitiveBinOp pmod = new PrimitiveBinOp("`%");
          global.setValue(pmod.getName(), pmod);

          PrimitiveBinOp pmult = new PrimitiveBinOp("`*");
          global.setValue(pmult.getName(), pmult);

          PrimitiveUnOp pneg = new PrimitiveUnOp("`!");
          global.setValue(pneg.getName(), pneg);

          PrimitiveVecBinOp pvplus = new
PrimitiveVecBinOp("`'+'");
```

```
          global.setValue(pvplus.getName(), pvplus);

          PrimitiveVecBinOp pvminus = new PrimitiveVecBinOp("`'-
'");
          global.setValue(pvminus.getName(), pvminus);

          PrimitiveVecBinOp pvand = new
PrimitiveVecBinOp("`'&'");
          global.setValue(pvand.getName(), pvand);

          PrimitiveVecBinOp pvor = new PrimitiveVecBinOp("`'|'");
          global.setValue(pvor.getName(), pvor);

          PrimitiveBinOp pand = new PrimitiveBinOp("`&");
          global.setValue(pand.getName(), pand);

          PrimitiveBinOp por = new PrimitiveBinOp("`|");
          global.setValue(por.getName(), por);

          PrimitiveVecBinOp pvxor = new
PrimitiveVecBinOp("`'^'");
          global.setValue(pvxor.getName(), pvxor);

          PrimitiveVecBinOp pvmult = new
PrimitiveVecBinOp("`'*'");
          global.setValue(pvmult.getName(), pvmult);

          PrimitiveVecBinOp pconcat = new
PrimitiveVecBinOp("`'++'");
          global.setValue(pconcat.getName(), pconcat);

          PrimitiveVecUnOp plsb = new PrimitiveVecUnOp("`'lsb'");
          global.setValue(plsb.getName(), plsb);

          PrimitiveVecUnOp pmsbs = new
PrimitiveVecUnOp("`'msbs'");
          global.setValue(pmsbs.getName(), pmsbs);

          PrimitiveVecUnOp plsbs = new
PrimitiveVecUnOp("`'lsbs'");
          global.setValue(plsbs.getName(), plsbs);

          PrimitiveVecUnOp pmsb = new PrimitiveVecUnOp("`'msb'");
          global.setValue(pmsb.getName(), pmsb);

          PrimitiveVecUnOp pvneg = new PrimitiveVecUnOp("`'~'");
          global.setValue(pvneg.getName(), pvneg);

          PrimitiveVecBinOp pvshleft = new
PrimitiveVecBinOp("`'<<'");
```

```
          global.setValue(pvshleft.getName(), pvshleft);

          PrimitiveVecBinOp pvshright = new
PrimitiveVecBinOp("`'>>'");
          global.setValue(pvshright.getName(), pvshright);

          PrimitiveVecBinOp pvlt = new PrimitiveVecBinOp("`'<'");
          global.setValue(pvlt.getName(), pvlt);

          PrimitiveVecBinOp pvgt = new PrimitiveVecBinOp("`'>'");
          global.setValue(pvgt.getName(), pvgt);

          PrimitiveVecBinOp pvleq = new
PrimitiveVecBinOp("`'<='");
          global.setValue(pvleq.getName(), pvleq);

          PrimitiveVecBinOp pvgeq = new
PrimitiveVecBinOp("`'>='");
          global.setValue(pvgeq.getName(), pvgeq);

          PrimitiveVecBinOp pveq = new
PrimitiveVecBinOp("`'=='");
          global.setValue(pveq.getName(), pveq);

          PrimitiveVecBinOp pvneq = new
PrimitiveVecBinOp("`'!='");
          global.setValue(pvneq.getName(), pvneq);

          PrimitiveVecBinOp pvbitsel = new
PrimitiveVecBinOp("`'");
          global.setValue(pvbitsel.getName(), pvbitsel);

          PrimitiveVecUnOp pwidth = new
PrimitiveVecUnOp("`'width'");
          global.setValue(pwidth.getName(), pwidth);

          PrimitiveVectorInput pvecin = new
PrimitiveVectorInput();
          global.setValue(pvecin.getName(), pvecin);

          PrimitiveVectorOutput pvecout = new
PrimitiveVectorOutput();
          global.setValue(pvecout.getName(), pvecout);

      } catch(DuplicationException e){
          if (Admin.debug){
            e.printStackTrace();
          }
          System.err.println("Duplicate primitive Symbol Table
entry");
```

```
          return null;
      }
      /* create rememberTable */
      rememberTable = new Vector();
      portDecl = new Vector();
      varDecl = new Vector();
      instructions = new Vector();
      return global;
    }

    public static void forceTable() throws ForceFailedException
    {
      /* force out the remember Table */

      int n = rememberTable.size();
      for (int i = 0; i < n; i++){
          Object t = rememberTable.get(i);
          if (t instanceof Thunk){
            ((Thunk)t).force(null);
          } else if (t instanceof Environment){
            ((Environment)t).force(null);
          } else {
            System.err.println("Cannot force: "+t);
            throw new ForceFailedException();
          }
      }

      if (fileName == null){
          System.err.println("Warning: no output file specified.
Using default name 'unnamedFile.v'");
          fileName = "unnamedFile.v";
      }
      try{
          PrintWriter outfile = new PrintWriter(new
FileWriter(fileName));
          if (moduleName == null){
            System.err.println("Warning: No module name
specified. Using default name 'UnnamedModule'");
            moduleName = "UnnamedModule";
          }
          outfile.print("module "+moduleName+"(");

          Collections.sort(portDecl);
          for (int i = 0; i<portDecl.size(); i++){
            int p = ((Declar)portDecl.get(i)).getPos();
            if (p != i+1){
                System.err.println("Warning: ["+i+"] Port
position badly declared:"+p);
            }
```

```
            outfile.print(portDecl.get(i)+((i == portDecl.size()
- 1)?"":", "));
          }
          outfile.println(");");
          outfile.println();
          for (int i = 0; i<portDecl.size(); i++){
            outfile.println(((Declar)portDecl.get(i)).declare());
          }
          outfile.println();
          for (int i = 0; i<varDecl.size(); i++){
            outfile.println(((Declar)varDecl.get(i)).declare());
          }
          outfile.println();
          for (int i = 0; i<instructions.size(); i++){
            outfile.println(instructions.get(i));
          }
          outfile.println();
          outfile.println("endmodule");
          outfile.close();
      }catch (IOException e){
          System.err.println("Unable to print to output file");
          throw new ForceFailedException();
      }
      if (genTestBench){
          if (benchName == null){
            System.err.println("Warning: no bench file specified.
Using default name 'unnamedBench.v'");
            benchName = "unnamedBench.v";
          }
          try{
            PrintWriter outfile = new PrintWriter(new
FileWriter(benchName));
            outfile.println("`include \""+fileName+"\"");
            outfile.println("module testbench;");
            for (int i = 0; i<portDecl.size(); i++){

outfile.println(((Declar)portDecl.get(i)).declareBench());
            }
            outfile.println();
            for (int i = 0; i<portDecl.size(); i++){
                int w =
((Declar)portDecl.get(i)).getInputWidth();
                String name =
((Declar)portDecl.get(i)).toString();
                if (w > 0){
                  outfile.println("integer "+name+"_iter ;");
                }
            }
            outfile.println();
            outfile.println("initial begin");
```

```
            String offset = "\t";
            for (int i = 0; i<portDecl.size(); i++){
                int w =
((Declar)portDecl.get(i)).getInputWidth();
                String name =
((Declar)portDecl.get(i)).toString();
                if (w > 0){
                  outfile.println(offset+"for ("+name+"_iter = 0;
"+name+"_iter < "+(1<<w)+"; "+name+"_iter = "+name+"_iter +1)
begin");
                  offset += "\t";
                }
            }
            for (int i = 0; i<portDecl.size(); i++){
                int w =
((Declar)portDecl.get(i)).getInputWidth();
                String name =
((Declar)portDecl.get(i)).toString();
                if (w > 0){
                  outfile.println(offset+name+" = "+name+"_iter
;");
                }
            }
            outfile.println(offset+"#1;");
            outfile.print(offset+"$display(\"");
            for (int i = 0; i<portDecl.size(); i++){
                int w =
((Declar)portDecl.get(i)).getInputWidth();
                String name =
((Declar)portDecl.get(i)).toString();
                if (w > 0){
                  outfile.print("%h ");
                }
            }
            outfile.print("-> ");
            for (int i = 0; i<portDecl.size(); i++){
                int w =
((Declar)portDecl.get(i)).getInputWidth();
                String name =
((Declar)portDecl.get(i)).toString();
                if (w == -1){
                  outfile.print("%h ");
                }
            }
            outfile.print("\"");
            for (int i = 0; i<portDecl.size(); i++){
                int w =
((Declar)portDecl.get(i)).getInputWidth();
                String name =
((Declar)portDecl.get(i)).toString();
```

```
                if (w > 0){
                  outfile.print(", "+name);
                }
            }
            for (int i = 0; i<portDecl.size(); i++){
                int w =
((Declar)portDecl.get(i)).getInputWidth();
                String name =
((Declar)portDecl.get(i)).toString();
                if (w == -1){
                  outfile.print(", "+name);
                }
            }
            outfile.println(");");
            for (int i = 0; i<portDecl.size(); i++){
                int w =
((Declar)portDecl.get(i)).getInputWidth();
                String name =
((Declar)portDecl.get(i)).toString();
                if (w > 0){
                  outfile.print("end ");
                }
            }
            outfile.println("end");
            outfile.print(moduleName+" my"+moduleName+"(");
            for (int i = 0; i<portDecl.size(); i++){
                String name =
((Declar)portDecl.get(i)).toString();
                outfile.print(name+((i+1 ==
portDecl.size())?"":", "));
            }
            outfile.println(");");
            outfile.println("endmodule");
            outfile.close();
          }catch (IOException e){
            System.err.println("Unable to print to test bench
file");
            throw new ForceFailedException();
          }
      }
    }

    public static void printForceTable()
    {
      for (int i = 0; i<rememberTable.size(); i++){
          System.out.println("["+i+"]:"+rememberTable.get(i));
      }
    }
```

```
    public static void setFileName(String fn) throws
DuplicationException
    {
      if (fileName != null){
          throw new DuplicationException();
      }
      fileName = fn;
    }


    public static void setBenchName(String bn) throws
DuplicationException
    {
      if (benchName != null){
          throw new DuplicationException();
      }
      benchName = bn;
    }

    public static void setModuleName(String mn) throws
DuplicationException, BadVerilogException
    {
      if (moduleName != null){
          throw new DuplicationException();
      }
      Declar.validate(mn);
      moduleName = mn;
    }
}
```

A.2.2 ApplyFailedException.java

```
package LCSL;

/**
 * Thrown when applying a primitive fails
 *
 * @author Bogdan Caprita - bc2008@columbia.edu
 */

public class ApplyFailedException extends Exception
{
}
```

A.2.3 Assignment.java

```
package LCSL;

/**
```

```
 * LCLS arrow assignment
 *
 * @author Bogdan Caprita - bc2008@columbia.edu
 */

public class Assignment implements Expression
{
    private HierarchicalVariable assignee; // can be var OR
sys1.sys2.sys3.var
    private Expression assigned; // the right hand side

    public Assignment() {}

    public Assignment(HierarchicalVariable v, Expression e){
      assignee = v;
      assigned = e;
    }

    public STEntry eval(Environment env) throws
EvalFailedException
    {
      /* Find where the hierarchical variable points to */
      Environment containingEnv = null;
      try{
          containingEnv = assignee.resolve(env);
      }catch(UndefinedException e){
          System.err.println("Undefined ST entry: "+assignee);
          throw new EvalFailedException();
      }
      STEntry entry = assigned.eval(env); // eval the right hand
side
      try{

containingEnv.setValue(assignee.getVariable().getName(), entry);
      }catch(DuplicationException e){
          System.err.println("Assign: Tried to duplicate ST
entry: "+assignee.getVariable().getName());
          throw new EvalFailedException();
      }
      return entry;
    }

    public void print(String offset)
    {
      System.out.println(offset+"Assignment");
      assignee.print(offset+OFFSET);
      assigned.print(offset+OFFSET);
    }

}
```

A.2.4 BadVerilogException.java

```
package LCSL;

/**
 * Thrown when incorrect verilog might be generated
 *
 * @author Bogdan Caprita - bc2008@columbia.edu
 */

public class BadVerilogException extends Exception
{
}
```

A.2.5 BitVector.java

```
package LCSL;

/**
 * The LCLS literal bit vector data type (string of 0's and 1's)
 *
 * @author Bogdan Caprita - bc2008@columbia.edu
 */

public class BitVector extends BitWire
{
    public BitVector(String v, int w)
    {
      super(v, w);
      if (v == null){
          width = w;
          value = "";
          for (int i = 0; i<w; i++){
            value += "0";
          }
      }
    }

    public String getValue()
    {
      return width+"'b"+value;
    }
}
```

A.2.6 BitWire.java

```
package LCSL;

/**
```

```
 * Generic LCLS Vector datatype.
 * can be either a BitVector (literal string of 0's and 1's) or
 * a Wire (similar to the Verilog wire data type)
 *
 * @author Bogdan Caprita - bc2008@columbia.edu
 */

public abstract class BitWire implements Expression, STEntry
{
    int width;
    String value;

    public BitWire(String value, int w)
    {
      this.value = value;
      width = w;
    }

    public String getValue()
    {
      return value;
    }

    public int getWidth()
    {
      return width;
    }

    public STEntry eval(Environment env)
    {
      return this;
    }

    public STEntry force(Environment env)
    {
      return this;
    }

    public boolean fullyEvaled()
    {
      return true;
    }

    public void print(String offset)
    {
      System.out.println(offset+value+"["+width+"]");
    }
}
```

A.2.7 Bool.java

```
package LCSL;

/**
 * LCLS boolean data type
 *
 * @author Bogdan Caprita - bc2008@columbia.edu
 */

public class Bool implements Expression, STEntry
{
    boolean value;

    public Bool(boolean value)
    {
      this.value = value;
    }

    public STEntry eval(Environment env)
    {
      return this;
    }


    public STEntry force(Environment env)
    {
      return this;
    }

    public boolean getValue()
    {
      return value;
    }

    public boolean fullyEvaled()
    {
      return true;
    }

    public void print(String offset)
    {
      System.out.println(offset+value);
    }
}
```

A.2.8 Component.java

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

```
/**
 * User defined LCLS component.
 *
 * @author Bogdan Caprita - bc2008@columbia.edu
 */

public class Component extends CompPrim implements Expression
{
    private Body body; // body of component
    private Environment defEnv; // env where comp was defined

    public Component() {}

    public Component(Vector a, Vector b, Body c) {
      portsIn = new Variable[a.size()];
      for (int i = 0; i<portsIn.length; i++){
          portsIn[i] = (Variable)a.get(i);
      }
      portsOut = new Variable[b.size()];
      for (int i = 0; i<portsOut.length; i++){
          portsOut[i] = (Variable)b.get(i);
      }
      body = c;
    }

    public STEntry eval(Environment env)
    {
      defEnv = env;
      return this;
    }

    public Body getBody()
    {
      return body;
    }

    public Environment getEnv()
    {
      return defEnv;
    }

    public void print(String offset)
    {
      System.out.println(offset+"Component: "+name);
      int i ;
      System.out.print(offset+OFFSET);
      if (portsIn == null){
          System.out.println("No input ports");
      }else{
          System.out.print("In ports: ");
```

```
            for (i = 0; i<portsIn.length - 1; i++)
              System.out.print(portsIn[i] + ".");
            System.out.println(portsIn[i]);
        }
        System.out.print(offset+OFFSET);
        if (portsOut == null){
            System.out.println("No output ports");
        }else{
            System.out.print("Out ports: ");
            for (i = 0; i<portsOut.length - 1; i++)
              System.out.print(portsOut[i] + ".");
            System.out.println(portsOut[i]);
        }

        body.print(offset+OFFSET);
    }
}
```

A.2.9 CompPrim.java

```
package LCSL;

/**
 * Generic LCLS Component. Can be either a Primitive (component
 * built in the language), or Component (user defined)
 *
 * @author Bogdan Caprita - bc2008@columbia.edu
 */

public abstract class CompPrim implements STEntry
{

    protected Variable[] portsIn, portsOut; // formal names of
arguments
    protected String name;

    public boolean fullyEvaled() {
      return true;
    }

    public Variable[] getPortsIn()
    {
      return portsIn;
    }

    public Variable[] getPortsOut()
    {
      return portsOut;
    }
```

```
    public String getName()
    {
      return name;
    }

    public STEntry force(Environment env)
    {
      System.err.println("Forced CompPrim - should not happen!");
      return this;
    }

}
```

A.2.10 DuplicationException.java

```
package LCSL;

/**
 * Thrown when an attempt is made to overwrite an existing ST
entry.
 * In LCLS, ST entries are bound at most once
 *
 * @author Bogdan Caprita - bc2008@columbia.edu
 */

public class DuplicationException extends Exception
{
}
```

A.2.11 Environment.java

```
package LCSL;

import java.util.*;


/**
 * The evaluation and forcing environment.
 * Functions as ST (Symbol Table) and is organized in a hierarchy
of frames.
 *
 * @author Bogdan Caprita - bc2008@columbia.edu
 */

public class Environment implements STEntry
{
    private Hashtable frame;
    private Environment parentEnv;
    private CompPrim compprim;
    private String name; // for naming purposes
```

```
    private int numChildren; // for naming purposes
    private Vector children; // for debugging
    private static STEntry NIL = new Environment("NIL");

    public boolean fullyEvaled()
    {
      System.err.println("Called fullyEvaled on "+name);
      return true;
    }

    public Environment(String n)
    {
      frame = new Hashtable();
      parentEnv = null;
      compprim = null;
      name = n;
      numChildren = 0;
      children = new Vector();
    }

    protected void finalize()
    {
      parentEnv.removeChild(this);
    }


    /* returns null if the name is found in the ST, but no entry
is defined.
       Throws exception if name doesn't show up in ST */
    public STEntry lookupValue(String name) throws
UndefinedException
    {
      STEntry value;
      try {
          int pos = Integer.parseInt(name);
          if (compprim == null){
            System.err.println("Tried to reference port of
inexisting component");
            throw new UndefinedException();
          }
          Variable[] pin = compprim.getPortsIn();
          int l1;
          if (pin == null) {
            l1 = 0;
          } else {
            l1 = pin.length;
          }
          if (pos <= l1){
            name = pin[pos-1].getName();
          } else {
```

```
            Variable[] pout = compprim.getPortsOut();
            name = pout[pos-l1-1].getName();
          }
      } catch (NumberFormatException e) {
      } catch (ArrayIndexOutOfBoundsException e){
          System.err.println("No port numbered: "+name);
          throw new UndefinedException();
      }

      if ((value = (STEntry)frame.get(name)) == NIL){
          return null;
      }
      if (value != null){
          return value;
      }
      if (parentEnv != null){
          return parentEnv.lookupValue(name);
      }
      throw new UndefinedException();
    }

    public void addEntry(String name) throws DuplicationException
    {
      if (frame.get(name) != null){
          throw new DuplicationException();
      }
      frame.put(name, NIL);
    }

    public void setValue(String name, STEntry entry) throws
DuplicationException
    {
      try {
          int pos = Integer.parseInt(name);
          if (compprim == null){
            System.err.println("Tried to reference port of
inexisting component");
            throw new DuplicationException();
          }
          Variable[] pin = compprim.getPortsIn();
          int l1;
          if (pin == null) {
            l1 = 0;
          } else {
            l1 = pin.length;
          }
          if (pos <= l1){
            name = pin[pos-1].getName();
          } else {
            Variable[] pout = compprim.getPortsOut();
```

```
            name = pout[pos-l1-1].getName();
          }
      } catch (NumberFormatException e) {
      } catch (ArrayIndexOutOfBoundsException e){
          System.err.println("No port numbered: "+name);
          throw new DuplicationException();
      }


      STEntry value = (STEntry)frame.get(name);

      if (value != null && value != NIL){
          System.err.println("Duplication detected when assigning
"+name+" = "+value);
          throw new DuplicationException();
      }
      frame.put(name, entry);

    }

    public void setParent(Environment env)
    {
      parentEnv = env;
      name = "Env"+env.getNumChildren();
      env.addChild(this);
    }


    public void setCompPrim(CompPrim cp)
    {
      this.compprim = cp;
    }

    public CompPrim getCompPrim()
    {
      return compprim;
    }

    public String getHierarchicalName()
    {
      return ((parentEnv ==
null)?"":parentEnv.getHierarchicalName()+"_") + name;
    }


    public void addChild(Environment childEnv)
    {
      numChildren ++;
      children.add(childEnv);
    }
```

```
    public void removeChild(Environment childEnv)
    {
      numChildren --;
      children.remove(childEnv);
    }

    public int getNumChildren()
    {
      return numChildren;
    }

    public STEntry force(Environment env) throws
ForceFailedException
    {
      if (name.compareTo("global") != 0){
          if (compprim == null){
            System.err.println("Cannot force: no component
defined");
            throw new ForceFailedException();
          }
          if (compprim instanceof Primitive){
            try{
                ((Primitive)compprim).apply(this);
            }catch(ApplyFailedException e){
                System.err.println("Failed applying the
primitive");
                if (Admin.debug){
                  e.printStackTrace();
                }
                throw new ForceFailedException();
            }
          } else {
            // it's a user defined component
            Component c = (Component)compprim;
            try {
                c.getBody().eval(this);
            } catch (EvalFailedException e){
                System.err.println("Failed evaluating the
component body");
                if (Admin.debug){
                  e.printStackTrace();
                }
                throw new ForceFailedException();
            }
          }
      }else{
          System.err.println("Global environment should not be
forced");
      }
```

```
      return null;
    }

    public static final String OFFSET = " . ";
    public void print(String offset)
    {
      System.out.println(offset+"<ENV:"+getHierarchicalName()+">"
);

      for (Enumeration e = frame.keys() ; e.hasMoreElements() ;)
{
          String key = (String)e.nextElement();
          String value;
          System.out.println(offset+ "\""+key+"\"-
>"+frame.get(key));

      }
      System.out.println();
      for (int i = 0; i<children.size(); i++)
          ((Environment)children.get(i)).print(offset+OFFSET);
    }

    public String toString()
    {
      return "Environment <"+getHierarchicalName()+">";
    }

}
```

A.2.12 EvalFailedException.java

```
package LCSL;

/**
 * Thrown when the evaluation fails for an Expression
 *
 * @author Bogdan Caprita - bc2008@columbia.edu
 */

public class EvalFailedException extends Exception
{
}
```

A.2.13 Expression.java

```
package LCSL;

/**
 * Interface for an LCLS expression
 *
```

```
 * The basic node in the LCLS tree
 *
 * @author Bogdan Caprita - bc2008@columbia.edu
 */

public interface Expression
{
    public final String OFFSET = " _ ";

    public STEntry eval(Environment env) throws
EvalFailedException;

    public void print(String offset);
}
```

A.2.14 ForceFailedException.java

```
package LCSL;

/**
 * Thrown when forcing cannot go through
 *
 * @author Bogdan Caprita - bc2008@columbia.edu
 */

public class ForceFailedException extends Exception
{
}
```

A.2.15 HierarchicalVariable.java

```
package LCSL;

/**
 * The LCLS variable resolution
 *
 * In LCLS, a variable can be referenced as
Sys0_Sys1.Sys2.Sys3.var where
 * each Sys_i is a port name in system Sys_(i-1) and represents a
system
 * as well. var is the port name where the variable of interest
is located
 * Alternatively, Sys_i and var can be replaced by port numbers (
for i > 0 )
 *
 * @author Bogdan Caprita - bc2008@columbia.edu
 */

public class HierarchicalVariable implements Expression
{
```

```
    private Variable[] names;

    public HierarchicalVariable() {}
    public HierarchicalVariable(String s){
      names = new Variable[1];
      names[0] = new Variable(s);
    }

    public HierarchicalVariable(Variable[] v){
      names = v;
    }

    public HierarchicalVariable(HierarchicalVariable hv, String
last)
    {
      this(hv, new Variable(last));
    }

    public HierarchicalVariable(HierarchicalVariable hv, Variable
last)
    {
      Variable[] old_names = hv.names;
      names = new Variable[old_names.length + 1];
      for (int i = 0; i<old_names.length; i++){
          names[i] = old_names[i];
      }
      names[old_names.length] = last;
    }

    public STEntry eval(Environment env) throws
EvalFailedException
    {
      Environment thunkenv = null;
      try {
          thunkenv = resolve(env);
      }catch (UndefinedException e){
          System.err.println("Could not resolve: "+toString());
          throw new EvalFailedException();
      }
      return new Thunk(getVariable(), thunkenv);
    }

    public Environment resolve(Environment env) throws
UndefinedException
    {
      Environment parent = env;
      for (int i = 0; i < names.length - 1; i++){
          STEntry entry = parent.lookupValue(names[i].getName());
          if (!( entry instanceof Environment )){
```

```
            System.err.println("Invalid hierarchical name:
"+toString());
            throw new UndefinedException();
          }
          parent = (Environment)entry;
      }
      return parent;
    }

    public Variable getVariable()
    {
      return names[names.length-1];
    }

    public boolean fullyEvaled()
    {
      return false;
    }


    public String toString()
    {
      String ret = "";
      int i;
      for (i = 0; i<names.length - 1; i++){
          ret += names[i].getName()+".";
      }
      ret += names[i].getName();
      return ret;
    }

    public void print(String offset)
    {
      System.out.println(offset+"HVar: "+toString());
    }
}
```

A.2.16 IfStatement.java

```
package LCSL;

/**
 * The LCLS if statement
 *
 * @author Bogdan Caprita - bc2008@columbia.edu
 */

public class IfStatement implements Expression
{
    private Expression predicate;
```

```
    private Body ifCase, elseCase;

    public IfStatement(Expression exp, Body b1, Body b2)
    {
      predicate = exp;
      ifCase = b1;
      elseCase = b2;
    }

    public STEntry eval(Environment env) throws
EvalFailedException
    {
      STEntry predicateTruth;
      try{
          predicateTruth = predicate.eval(env).force(env);
      } catch (EvalFailedException e){
          System.err.println("If: eval of predicate failed");
          throw new EvalFailedException();
      }catch (ForceFailedException e){
          System.err.println("If: force of predicate failed");
          throw new EvalFailedException();
      }
      boolean ifBranch;
      if (predicateTruth instanceof Bool){
          ifBranch = ((Bool)predicateTruth).getValue();
      }else{
          System.err.println("If: predicate is not a boolean");
          throw new EvalFailedException();
      }
      if (ifBranch){
          return ifCase.eval(env);
      }else{
          return elseCase.eval(env);
      }
    }

    public void print(String offset)
    {
      System.out.println(offset+"if Statement:");
      predicate.print(offset+OFFSET);
      ifCase.print(offset+OFFSET);
      elseCase.print(offset+OFFSET);
    }
}
```

A.2.17 NameGenerator.java

```
package LCSL;

public class NameGenerator {
```

```
    public static int nextNumber = 0;

    public NameGenerator() {
    }

    public int getNextNumber() {
      return nextNumber++;
    }
}
```

A.2.18 Number.java

```
package LCSL;

/**
 * LCLS integer data type
 *
 * @author Bogdan Caprita - bc2008@columbia.edu
 */

public class Number implements Expression, STEntry
{
    int value;

    public Number(int value)
    {
      this.value = value;
    }

    public STEntry eval(Environment env)
    {
      return this;
    }


    public STEntry force(Environment env)
    {
      return this;
    }

    public int getValue()
    {
      return value;
    }

    public boolean fullyEvaled()
    {
      return true;
    }
```

```
    public void print(String offset)
    {
      System.out.println(offset+value);
    }
}
```

A.2.19 PrimitiveBinOp.java

```
package LCSL;

/**
 * Primitive binary operators on integers and booleans
 *
 * @author Bogdan Caprita - bc2008@columbia.edu
 */

public class PrimitiveBinOp extends Primitive
{

    public PrimitiveBinOp(String name)
    {
      this.name = name;
      requiresForce = false;
      portsIn = createPortsIn();
      portsOut = createPortsOut();
    }

    public void apply(Environment env) throws
ApplyFailedException
    {
      STEntry A, B, output;

      try{
          A = (env.lookupValue("A")).force(env);
          B = (env.lookupValue("B")).force(env);
      } catch(UndefinedException e) {
          System.err.println("Cannot get inputs to "+name);
          throw new ApplyFailedException();
      }catch(ForceFailedException e) {
          System.err.println("Cannot get inputs to "+name);
          throw new ApplyFailedException();
      }

      if (name.compareTo("`==") == 0){
          if (A instanceof Number){
            if (B instanceof Number){
                if ( ((Number)A).getValue() ==
((Number)B).getValue() ){
                  output = new Bool(true);
```

```
                }else{
                  output = new Bool(false);
                }
            } else {
                System.err.println("Incompatible types in
equality");
                throw new ApplyFailedException();
            }
          } else if (A instanceof Bool){
            if (B instanceof Bool){
                if ( ((Bool)A).getValue() == ((Bool)B).getValue()
){
                  output = new Bool(true);
                }else{
                  output = new Bool(false);
                }
            } else {
                System.err.println("Incompatible types in
equality");
                throw new ApplyFailedException();
            }
          } else {
            System.err.println("Invalid types in equality");
            throw new ApplyFailedException();
          }
      }


      else if (name.compareTo("`!=") == 0){
          if (A instanceof Number){
            if (B instanceof Number){
                if ( ((Number)A).getValue() !=
((Number)B).getValue() ){
                  output = new Bool(true);
                }else{
                  output = new Bool(false);
                }
            } else {
                System.err.println("Incompatible types in !=");
                throw new ApplyFailedException();
            }
          } else if (A instanceof Bool){
            if (B instanceof Bool){
                if ( ((Bool)A).getValue() != ((Bool)B).getValue()
){
                  output = new Bool(true);
                }else{
                  output = new Bool(false);
                }
            } else {
```

```
                System.err.println("Incompatible types in !=");
                throw new ApplyFailedException();
            }
          } else {
            System.err.println("Invalid types in !=");
            throw new ApplyFailedException();
          }
      }


      else if (name.compareTo("`<") == 0){
          if (A instanceof Number){
            if (B instanceof Number){
                if ( ((Number)A).getValue() <
((Number)B).getValue() ){
                  output = new Bool(true);
                }else{
                  output = new Bool(false);
                }
            } else {
                System.err.println("Incompatible types in <");
                throw new ApplyFailedException();
            }
          } else {
            System.err.println("Invalid types in <");
            throw new ApplyFailedException();
          }
      }


      else if (name.compareTo("`>") == 0){
          if (A instanceof Number){
            if (B instanceof Number){
                if ( ((Number)A).getValue() >
((Number)B).getValue() ){
                  output = new Bool(true);
                }else{
                  output = new Bool(false);
                }
            } else {
                System.err.println("Incompatible types in >");
                throw new ApplyFailedException();
            }
          } else {
            System.err.println("Invalid types in >");
            throw new ApplyFailedException();
          }
      }
```

```
      else if (name.compareTo("`<=") == 0){
          if (A instanceof Number){
            if (B instanceof Number){
                if ( ((Number)A).getValue() <=
((Number)B).getValue() ){
                  output = new Bool(true);
                }else{
                  output = new Bool(false);
                }
            } else {
                System.err.println("Incompatible types in <=");
                throw new ApplyFailedException();
            }
          } else {
            System.err.println("Invalid types in <=");
            throw new ApplyFailedException();
          }
      }

      else if (name.compareTo("`>=") == 0){
          if (A instanceof Number){
            if (B instanceof Number){
                if ( ((Number)A).getValue() >=
((Number)B).getValue() ){
                  output = new Bool(true);
                }else{
                  output = new Bool(false);
                }
            } else {
                System.err.println("Incompatible types in >=");
                throw new ApplyFailedException();
            }
          } else {
            System.err.println("Invalid types in >=");
            throw new ApplyFailedException();
          }
      }

      else if ((name.compareTo("`+") == 0)){
          if ((A instanceof Number) && (B instanceof Number)){
            output = new Number(((Number)A).getValue() +
((Number)B).getValue() );
          } else {
            System.err.println("Incompatible types for
addition");
            throw new ApplyFailedException();
          }
      }

      else if ((name.compareTo("`-") == 0)){
```

```
          if ((A instanceof Number) && (B instanceof Number)){
            output = new Number(((Number)A).getValue() -
((Number)B).getValue() );
          } else {
            System.err.println("Incompatible types for
subtraction");
            throw new ApplyFailedException();
          }
      }

      else if ((name.compareTo("`/") == 0)){
          if ((A instanceof Number) && (B instanceof Number)){
            if (((Number)B).getValue() == 0){
                System.err.println("Error: Division by 0");
                throw new ApplyFailedException();
            }
            output = new Number(((Number)A).getValue() /
((Number)B).getValue() );
          } else {
            System.err.println("Incompatible types for
division");
            throw new ApplyFailedException();
          }
      }

      else if ((name.compareTo("`%") == 0)){
          if ((A instanceof Number) && (B instanceof Number)){
            if (((Number)B).getValue() == 0){
                System.err.println("Error: Division by 0");
                throw new ApplyFailedException();
            }
            output = new Number(((Number)A).getValue() %
((Number)B).getValue() );
          } else {
            System.err.println("Incompatible types for mod
operation");
            throw new ApplyFailedException();
          }
      }

      else if ((name.compareTo("`*") == 0)){
          if ((A instanceof Number) && (B instanceof Number)){
            output = new Number(((Number)A).getValue() *
((Number)B).getValue() );
          } else {
            System.err.println("Incompatible types for
multiplication");
            throw new ApplyFailedException();
          }
      }
```

```
      else if ((name.compareTo("`&") == 0)){
          if ((A instanceof Bool) && (B instanceof Bool)){
            output = new Bool(((Bool)A).getValue() &&
((Bool)B).getValue() );
          } else {
            System.err.println("Incompatible types for boolean
AND");
            throw new ApplyFailedException();
          }
      }

      else if ((name.compareTo("`|") == 0)){
          if ((A instanceof Bool) && (B instanceof Bool)){
            output = new Bool(((Bool)A).getValue() ||
((Bool)B).getValue() );
          } else {
            System.err.println("Incompatible types for boolean
OR");
            throw new ApplyFailedException();
          }
      }


      else {
          System.err.println("Invalid binary operator");
          throw new ApplyFailedException();
      }
      try{
          env.setValue("X", output);
      } catch (DuplicationException e){
          System.err.println("Duplicate X");
          throw new ApplyFailedException();
      }
    }

    private Variable[] createPortsIn()
    {
      Variable[] inp = new Variable[2];
      inp[0] = new Variable("A");
      inp[1] = new Variable("B");
      return inp;
    }


    private Variable[] createPortsOut()
    {
      Variable[] outp = new Variable[1];
      outp[0] = new Variable("X");
      return outp;
```

```
    }

}
```

A.2.20 Primitive.java

```
package LCSL;

import java.util.*;

/**
 * Generic Primitive components (built in the language).
 *
 * @author Bogdan Caprita - bc2008@columbia.edu
 */

public abstract class Primitive extends CompPrim
{
    protected boolean requiresForce;

    public abstract void apply(Environment env) throws
ApplyFailedException;

    public Vector needToBeForced(Environment env)
    { /* will be implemented by those subclasses for which
       requiresForce is true */
      System.err.println("Default needToBeForced: Shouldn't get
called");
      return null;
    }

}
```

A.2.21 PrimitiveSet.java

```
package LCSL;

import java.util.*;

/**
 * Set built in primitive.
 *
 * @author Bogdan Caprita - bc2008@columbia.edu
 */

public class PrimitiveSet extends Primitive
{
    public PrimitiveSet()
    {
```

```
      portsIn = createPortsIn();
      portsOut = createPortsOut();
      name = "Set";
      requiresForce = true;
    }
    public void apply(Environment env) throws
ApplyFailedException
    {
      STEntry Name, Value;
      try{
          Name = (env.lookupValue("Name")).force(env);
          Value = (env.lookupValue("Value")).force(env);
      }catch(Exception e){
          System.err.println("Couldn't get one of Name, or
Value");
          throw new ApplyFailedException();
      }

      if (! (Name instanceof Str)){
          throw new ApplyFailedException();
      }
      String param = ((Str)Name).getValue();

      if (param.compareTo("FileName") == 0){
          if (! (Value instanceof Str)){
          throw new ApplyFailedException();
          }
          try{
            Admin.setFileName(((Str)Value).getValue());
          } catch (DuplicationException e){
            System.err.println("Duplicate set for file name");
            throw new ApplyFailedException();
          }
      } else if (param.compareTo("BuildName") == 0){
          if (! (Value instanceof Str)){
            throw new ApplyFailedException();
          }
          try{
            Admin.setModuleName(((Str)Value).getValue());
          } catch (DuplicationException e){
            System.err.println("Duplicate set for module name");
            throw new ApplyFailedException();
          } catch (BadVerilogException e){
            System.err.println("Invalid module name");
            throw new ApplyFailedException();
          }
      } else if (param.compareTo("GenTestBench") == 0){
          if (! (Value instanceof Bool)){
            throw new ApplyFailedException();
          }
```

```
            if (((Bool)Value).getValue()){
              Admin.genTestBench = true;
            }
        }

        else if (param.compareTo("BenchName") == 0){
            if (! (Value instanceof Str)){
              throw new ApplyFailedException();
            }
            try{
              Admin.setBenchName(((Str)Value).getValue());
            } catch (DuplicationException e){
              System.err.println("Duplicate set for bench name");
              throw new ApplyFailedException();
            }
        } else {
            System.err.println("Invalid name in Set");
            throw new ApplyFailedException();
        }

    }

    private Variable[] createPortsIn()
    {
      Variable[] inp = new Variable[2];
      inp[0] = new Variable("Name");
      inp[1] = new Variable("Value");
      return inp;
    }

    private Variable[] createPortsOut()
    {
      Variable[] outp = null;
      return outp;
    }

    public Vector needToBeForced(Environment env)
    {
      Vector v = new Vector();
      v.add(env);
      return v;
    }
}
```

A.2.22 PrimitiveUnOp.java

```
package LCSL;

/**
 * unary operator on booleans
```

```
 *
 * @author Bogdan Caprita - bc2008@columbia.edu
 */

public class PrimitiveUnOp extends Primitive
{

    public PrimitiveUnOp(String name)
    {
      this.name = name;
      requiresForce = false;
      portsIn = createPortsIn();
      portsOut = createPortsOut();
    }

    public void apply(Environment env) throws
ApplyFailedException
    {
      STEntry A, output;

      try{
          A = (env.lookupValue("A")).force(env);
      } catch(UndefinedException e) {
          System.err.println("Cannot get the input to "+name);
          throw new ApplyFailedException();
      }catch(ForceFailedException e) {
          System.err.println("Cannot get the input to "+name);
          throw new ApplyFailedException();
      }

      if (name.compareTo("`!") == 0){
          if (A instanceof Bool) {
            output = new Bool(!((Bool)A).getValue());
          } else {
            System.err.println("Incompatible type for boolean
negation");
            throw new ApplyFailedException();
          }
      }

      else {
          System.err.println("Invalid unary operator");
          throw new ApplyFailedException();
      }

      try{
          env.setValue("X", output);
      } catch (DuplicationException e){
          System.err.println("Duplicate X");
          throw new ApplyFailedException();
```

```
      }
    }


    private Variable[] createPortsIn()
    {
      Variable[] inp = new Variable[1];
      inp[0] = new Variable("A");
      return inp;
    }


    private Variable[] createPortsOut()
    {
      Variable[] outp = new Variable[1];
      outp[0] = new Variable("X");
      return outp;
    }

}
```

A.2.23 PrimitiveVecBinOp.java

```
package LCSL;

/**
 * Binary bit vector operators
 *
 * @author Bogdan Caprita - bc2008@columbia.edu
 */

public class PrimitiveVecBinOp extends Primitive
{

    public PrimitiveVecBinOp(String name)
    {
      this.name = name;
      requiresForce = false;
      portsIn = createPortsIn();
      portsOut = createPortsOut();
    }

    public void apply(Environment env) throws
ApplyFailedException
    {
      STEntry A, B;
      try{
          A = (env.lookupValue("A")).force(env);
          B = (env.lookupValue("B")).force(env);
      } catch(UndefinedException e) {
```

```
            System.err.println("Cannot get the inputs to "+name);
            throw new ApplyFailedException();
        }catch(ForceFailedException e) {
            System.err.println("Cannot get the inputs to "+name);
            throw new ApplyFailedException();
        }

        if (! (A instanceof BitWire)){
            System.err.println("Invalid operand to "+name);
            throw new ApplyFailedException();
        }


        int widthA = ((BitWire)A).getWidth();
        BitWire output;

        if ((name.compareTo("`'+'") == 0)||(name.compareTo("`'-'")
== 0) ||
            (name.compareTo("`'&'") == 0)||(name.compareTo("`'|'")
== 0) ||
            (name.compareTo("`'^'") == 0)
            ){
            if (! (B instanceof BitWire)){
              System.err.println("Invalid operand to "+name);
              throw new ApplyFailedException();
            }
            int widthB = ((BitWire)B).getWidth();

            if ( widthA != widthB){
              System.err.println("BitVecWires have diferent widths.
Cannot apply binary operator: "+name.substring(2,3));
              throw new ApplyFailedException();
            }

            output = new Wire(env.getHierarchicalName()+"_o1",
widthA);
            if (Admin.debug){
              System.out.println("(VERILOG:) wire ["+(widthA-
1)+":0] "+output.getValue()+";");
              System.out.println("(VERILOG:) assign "+
output.getValue()+
                                 " = "+((BitWire)A).getValue()+"
"+name.substring(2,3)+" "+((BitWire)B).getValue()+";");
            }
            try {
              Admin.addVariableDecl(output.getValue(), widthA);
            } catch (BadVerilogException e){
              System.err.println(output.getValue()+" is an invalid
indentifier");
              throw new ApplyFailedException();
```

```
          } catch (DuplicationException e){
            System.err.println(output.getValue()+" is a duplicate
indentifier");
            throw new ApplyFailedException();
          }
          Admin.addInstruction("assign "+ output.getValue()+
                          " = "+((BitWire)A).getValue()+"
"+name.substring(2,3)+" "+((BitWire)B).getValue()+";");
      }

      else if ((name.compareTo("`'==`'") == 0) ||
(name.compareTo("`'!=`'") == 0)||
             (name.compareTo("`'<=`'") == 0) ||
(name.compareTo("`'>=`'") == 0)||
             (name.compareTo("`'>`'") == 0) ||
(name.compareTo("`'<`'") == 0)
            ){
          if (! (B instanceof BitWire)){
            System.err.println("Invalid operand to "+name);
            throw new ApplyFailedException();
          }
          int widthB = ((BitWire)B).getWidth();

          int offs = 1;
          if ( (name.compareTo("`'>`'") == 0) ||
(name.compareTo("`'<`'") == 0) ){
            offs = 0;
          }
          if ( widthA != widthB){
            System.err.println("BitVecWires have diferent widths.
Cannot apply binary operator: "+ name.substring(2,3+offs));
            throw new ApplyFailedException();
          }

          output = new Wire(env.getHierarchicalName()+"_o1", 1);
          if (Admin.debug){
            System.out.println("(VERILOG:) wire
"+output.getValue()+";");
            System.out.println("(VERILOG:) assign "+
output.getValue()+
                           " = "+((BitWire)A).getValue()+"
"+name.substring(2,3+offs)+" "+((BitWire)B).getValue()+";");
          }
          try {
            Admin.addVariableDecl(output.getValue(), 1);
          } catch (BadVerilogException e){
            System.err.println(output.getValue()+" is an invalid
indentifier");
            throw new ApplyFailedException();
          } catch (DuplicationException e){
```

```
            System.err.println(output.getValue()+" is a duplicate
indentifier");
            throw new ApplyFailedException();
          }
          Admin.addInstruction("assign "+ output.getValue()+
                          " = "+((BitWire)A).getValue()+"
"+name.substring(2,3+offs)+" "+((BitWire)B).getValue()+";");
      }

      else if (name.compareTo("`'*'") == 0){
          if (! (B instanceof BitWire)){
            System.err.println("Invalid operand to "+name);
            throw new ApplyFailedException();
          }
          int widthB = ((BitWire)B).getWidth();

          output = new Wire(env.getHierarchicalName()+"_o1",
widthA+widthB);
          if (Admin.debug){
            System.out.println("(VERILOG:) wire
["+(widthA+widthB-1)+":0] "+output.getValue()+";");
            System.out.println("(VERILOG:) assign "+
output.getValue()+
                             " = "+((BitWire)A).getValue()+" *
"+((BitWire)B).getValue()+";");
          }
          try {
            Admin.addVariableDecl(output.getValue(),
widthA+widthB);
          } catch (BadVerilogException e){
            System.err.println(output.getValue()+" is an invalid
indentifier");
            throw new ApplyFailedException();
          }catch (DuplicationException e){
            System.err.println(output.getValue()+" is a duplicate
indentifier");
            throw new ApplyFailedException();
          }
          Admin.addInstruction("assign "+ output.getValue()+
                          " = "+((BitWire)A).getValue()+" *
"+((BitWire)B).getValue()+";");
      }

      else if (name.compareTo("`'++'") == 0){
          if (! (B instanceof BitWire)){
            System.err.println("Invalid operand to "+name);
            throw new ApplyFailedException();
          }
          int widthB = ((BitWire)B).getWidth();
```

```
          if (widthA == 0){
            output = (BitWire)B;
          } else if (widthB == 0){
            output = (BitWire)A;
          } else {
            output = new Wire(env.getHierarchicalName()+"_o1",
widthA+widthB);
            if (Admin.debug){
                System.out.println("(VERILOG:) wire
["+(widthA+widthB-1)+":0] "+output.getValue()+";");
                System.out.println("(VERILOG:) assign "+
output.getValue()+
                                   " = { "+((BitWire)A).getValue()+",
"+((BitWire)B).getValue()+" };");
            }
            try {
                Admin.addVariableDecl(output.getValue(),
widthA+widthB);
            } catch (BadVerilogException e){
                System.err.println(output.getValue()+" is an
invalid indentifier");
                throw new ApplyFailedException();
            }catch (DuplicationException e){
                System.err.println(output.getValue()+" is a
duplicate indentifier");
                throw new ApplyFailedException();
            }
            Admin.addInstruction("assign "+ output.getValue()+
                                 " = { "+((BitWire)A).getValue()+",
"+((BitWire)B).getValue()+" };");
          }
      }

      else if (name.compareTo("`'>>'") == 0){
          if (! (B instanceof Number)){
            System.err.println("Invalid operand to "+name);
            throw new ApplyFailedException();
          }
          int shift_amount = ((Number)B).getValue();
          if (shift_amount < 0){
            System.err.println("Shift amount cannot be
negative");
            throw new ApplyFailedException();
          }
          if (shift_amount > widthA){
            shift_amount = widthA;
          }
          if (shift_amount == 0){
            output = (BitWire)A;
          } else if (shift_amount == widthA){
```

```
            output = new BitVector(null, shift_amount);
          } else {
            output = new Wire(env.getHierarchicalName()+"_o1",
widthA);
            BitVector padding = new BitVector(null,
shift_amount);
            if (Admin.debug){
                System.out.println("(VERILOG:) wire ["+(widthA-
1)+":0] "+output.getValue()+";");
            }
            String instr =  "assign "+ output.getValue()+ " = {
"+
                padding.getValue();

            for (int i = widthA - 1; i >= shift_amount; i--){
                instr += ", "+((Wire)A).getValue()+"["+i+"]";
            }
            instr += " };";
            if (Admin.debug){
                System.out.println("(VERILOG:) "+instr);
            }
            try {
                Admin.addVariableDecl(output.getValue(), widthA);
            } catch (BadVerilogException e){
                System.err.println(output.getValue()+" is an
invalid indentifier");
                throw new ApplyFailedException();
            }catch (DuplicationException e){
                System.err.println(output.getValue()+" is a
duplicate indentifier");
                throw new ApplyFailedException();
            }
            Admin.addInstruction(instr);
          }
      }

      else if (name.compareTo("`'<<'") == 0){
          if (! (B instanceof Number)){
            System.err.println("Invalid operand to "+name);
            throw new ApplyFailedException();
          }
          int shift_amount = ((Number)B).getValue();
          if (shift_amount < 0){
            System.err.println("Shift amount cannot be
negative");
            throw new ApplyFailedException();
          }
          if (shift_amount > widthA){
            shift_amount = widthA;
          }
```

```
          if (shift_amount == 0){
            output = (BitWire)A;
          } else if (shift_amount == widthA){
            output = new BitVector(null, shift_amount);
          } else {
            output = new Wire(env.getHierarchicalName()+"_o1",
widthA);
            BitVector padding = new BitVector(null,
shift_amount);
            if (Admin.debug){
            System.out.println("(VERILOG:) wire ["+(widthA-
1)+":0] "+output.getValue()+";");
            }
            String instr =  "assign "+ output.getValue()+ " = {
";

            for (int i = widthA - shift_amount - 1; i >= 0; i--){
                instr += ((Wire)A).getValue()+"["+i+"], ";
            }
            instr += padding.getValue()+" };";
            if (Admin.debug){
                System.out.println("(VERILOG:) "+instr);
            }
            try {
                Admin.addVariableDecl(output.getValue(), widthA);
            } catch (BadVerilogException e){
                System.err.println(output.getValue()+" is an
invalid indentifier");
                throw new ApplyFailedException();
            }catch (DuplicationException e){
                System.err.println(output.getValue()+" is a
duplicate indentifier");
                throw new ApplyFailedException();
            }
            Admin.addInstruction(instr);
          }
      }


      else if (name.compareTo("`'") == 0){
          if (! (B instanceof Number)){
            System.err.println("Invalid operand to "+name);
            throw new ApplyFailedException();
          }
          int index = ((Number)B).getValue();
          if ((index < 0) || (index >= widthA)){
            System.err.println("Bit selection index out of
range");
            throw new ApplyFailedException();
          }
```

```
          output = new Wire(env.getHierarchicalName()+"_o1", 1);
          if (Admin.debug){
            System.out.println("(VERILOG:) wire ["+(widthA-
1)+":0] "+output.getValue()+";");
          }
          String instr =  "assign "+ output.getValue()+ " = " +
((Wire)A).getValue()+"["+index+"];";
          if (Admin.debug){
            System.out.println("(VERILOG:) "+instr);
          }
          try {
            Admin.addVariableDecl(output.getValue(), 1);
          } catch (BadVerilogException e){
            System.err.println(output.getValue()+" is an invalid
indentifier");
            throw new ApplyFailedException();
          }catch (DuplicationException e){
            System.err.println(output.getValue()+" is a duplicate
indentifier");
            throw new ApplyFailedException();
          }
          Admin.addInstruction(instr);
      }



      else {
          System.err.println("Unknown Binary Vector Operator: "+
name);
          throw new ApplyFailedException();
      }

      try{
          env.setValue("X", output);
      } catch (DuplicationException e){
          System.err.println("Duplicate X");
          throw new ApplyFailedException();
      }
    }

    private Variable[] createPortsIn()
    {
      Variable[] inp = new Variable[2];
      inp[0] = new Variable("A");
      inp[1] = new Variable("B");
      return inp;
    }


    private Variable[] createPortsOut()
```

```
    {
      Variable[] outp = new Variable[1];
      outp[0] = new Variable("X");
      return outp;
    }

}
```

A.2.24 PrimitiveVectorInput.java

```
package LCSL;

import java.util.*;

/**
 * Defines a vector input (which translates into a Verilog input
port)
 *
 * @author Bogdan Caprita - bc2008@columbia.edu
 */

public class PrimitiveVectorInput extends Primitive
{
    public PrimitiveVectorInput()
    {
      portsIn = createPortsIn();
      portsOut = createPortsOut();
      name = "VectorInput";
      requiresForce = true;
    }

    public void apply(Environment env) throws
ApplyFailedException
    {
      STEntry Name, Num, Width;
      try{
          Name = (env.lookupValue("Name")).force(env);
          Num = (env.lookupValue("Position")).force(env);
          Width = (env.lookupValue("Width")).force(env);
      }catch(Exception e){
          System.err.println("Couldn't get one of Name, Num, or
Width");
          throw new ApplyFailedException();
      }

      if (! (Name instanceof Str)){
          System.err.println(Name+" is not a String");
          throw new ApplyFailedException();
      }
      if (! (Num instanceof Number)){
```

```
            System.err.println(Num+" is not a Number");
            throw new ApplyFailedException();
        }
        if (! (Width instanceof Number)){
            System.err.println(Width+" is not a String");
            throw new ApplyFailedException();
        }
        String name = ((Str)Name).getValue()+"_i";
        int width = ((Number)Width).getValue();
        Wire output = new
Wire(/*env.getHierarchicalName()+"_i1"*/name, width);
        if (Admin.debug){
            System.out.println("(VERILOG:) input ["+(width-1)+":0]
"+ output.getValue()+
                               "; in position #
"+((Number)Num).getValue());
        }
        try {
            Admin.addInPortDecl(name, width,
((Number)Num).getValue());
        } catch (BadVerilogException e){
            System.err.println(name+" is an invalid port
indentifier");
            throw new ApplyFailedException();
        } catch (DuplicationException e){
            System.err.println(name+" is a duplicate port
indentifier");
            throw new ApplyFailedException();
        }
        try{
            env.setValue("Vector", output);
        } catch(DuplicationException e){
            System.err.println("Couldn't set Vec");
            throw new ApplyFailedException();
        }
    }

    private Variable[] createPortsIn()
    {
      Variable[] inp = new Variable[3];
      inp[0] = new Variable("Name");
      inp[1] = new Variable("Position");
      inp[2] = new Variable("Width");
      return inp;
    }


    private Variable[] createPortsOut()
    {
      Variable[] outp = new Variable[1];
```

```
      outp[0] = new Variable("Vector");
      return outp;
    }

    public Vector needToBeForced(Environment env)
    {
      Vector v = new Vector();
      v.add(new Thunk(portsOut[0], env));
      return v;
    }

}
```

A.2.25 PrimitiveVectorOutput.java

```
package LCSL;

import java.util.*;

/**
 * defines an output vector, which translates into an output
Verilog port
 *
 * @author Bogdan Caprita - bc2008@columbia.edu
 */

public class PrimitiveVectorOutput extends Primitive
{

    public PrimitiveVectorOutput()
    {
      portsIn = createPortsIn();
      portsOut = createPortsOut();
      name = "VectorOutput";
      requiresForce = true;
    }

    public void apply(Environment env) throws
ApplyFailedException
    {
      STEntry Name, Num;
      try{
          Name = (env.lookupValue("Name")).force(env);
          Num = (env.lookupValue("Position")).force(env);
      }catch(Exception e){
          System.err.println("Couldn't get Name or Num");
          throw new ApplyFailedException();
      }

      if (! (Name instanceof Str)){
```

```
            System.err.println(Name+" is not a string");
            throw new ApplyFailedException();
        }
        if (! (Num instanceof Number)){
            System.err.println(Num+" is not a number");
            throw new ApplyFailedException();
        }
        STEntry binding;
        try{
            binding = (env.lookupValue("Vector")).force(env);
        }catch(Exception e){
            System.err.println("Couldn't get Vector");
            throw new ApplyFailedException();
        }
        if (! (binding instanceof BitWire)){
            System.err.println(binding+" is not a BitWire");
            throw new ApplyFailedException();
        }
        String name = ((Str)Name).getValue()+"_o";
        int width = ((BitWire)binding).getWidth();
        Wire output = new
Wire(/*env.getHierarchicalName()+"_o1"*/name, width);
        if (Admin.debug){
            System.out.println("(VERILOG:) output "+((width ==
1)?"":("["+(width-1)+":0] "))+ output.getValue()+"; in position #
"+((Number)Num).getValue());
            System.out.println("(VERILOG:) assign
"+output.getValue()+" = "+((BitWire)binding).getValue()+";");
        }
        try {
            Admin.addOutPortDecl(name, width,
((Number)Num).getValue());
        } catch (BadVerilogException e){
            System.err.println(name+" is an invalid port
indentifier");
            throw new ApplyFailedException();
        } catch (DuplicationException e){
            System.err.println(name+" is a duplicate port
indentifier");
            throw new ApplyFailedException();
        }
        Admin.addInstruction("assign "+name+" =
"+((BitWire)binding).getValue()+";");
    }

    private Variable[] createPortsIn()
    {
      Variable[] inp = new Variable[3];
      inp[0] = new Variable("Name");
      inp[1] = new Variable("Position");
```

```
      inp[2] = new Variable("Vector");
      return inp;
    }


    private Variable[] createPortsOut()
    {
      Variable[] outp = null;
      return outp;
    }


    public Vector needToBeForced(Environment env)
    {
      Vector v = new Vector();
      v.add(env);
      return v;
    }

}
```

A.2.26 PrimitveVecUnOp.java

```
package LCSL;

/**
 * bit vector unary operators
 *
 * @author Bogdan Caprita - bc2008@columbia.edu
 */

public class PrimitiveVecUnOp extends Primitive
{

    public PrimitiveVecUnOp(String name)
    {
      this.name = name;
      requiresForce = false;
      portsIn = createPortsIn();
      portsOut = createPortsOut();
    }

    public void apply(Environment env) throws
ApplyFailedException
    {
      STEntry A;
      try{
          A = (env.lookupValue("A")).force(env);
      } catch(UndefinedException e) {
          System.err.println("Cannot get the input to "+name);
```

```
            throw new ApplyFailedException();
        }catch(ForceFailedException e) {
            System.err.println("Cannot get the input to "+name);
            throw new ApplyFailedException();
        }

        if (! (A instanceof BitWire)){
            throw new ApplyFailedException();
        }
        int width = ((BitWire)A).getWidth();

        STEntry output;

        /* this is where the differentiation is made among the
           various vector operators */

        if ((name.compareTo("`'lsb'") == 0) ||
(name.compareTo("`'msb'") == 0)){
            if (width > 1) {
              Wire out = new Wire(env.getHierarchicalName()+"_o1",
1);
              if (Admin.debug){
                  System.out.println("(VERILOG:) wire
"+out.getValue()+";");
                  System.out.println("(VERILOG:) assign "+
out.getValue()+
                                     " = "+((Wire)A).getValue()+"["+

((name.compareTo("`'msb'")==0)?""+(width-1):"0")
                                     +"];");
              }
              try{
                  Admin.addVariableDecl(out.getValue(), 1);
              } catch (BadVerilogException e){
                  System.err.println(out.getValue()+" is an invalid
indentifier");
                  throw new ApplyFailedException();
              } catch (DuplicationException e){
                  System.err.println(out.getValue()+" is a
duplicate indentifier");
                  throw new ApplyFailedException();

              }
              Admin.addInstruction("assign "+out.getValue() +
                                   " = "+((Wire)A).getValue()+"["+

((name.compareTo("`'msb'")==0)?""+(width-1):"0")
                                   +"];");
              output = out;
            } else if (width == 1){
```

```
            output = A;
          } else {
            BitVector out = new BitVector("'", 0);
            output = out;
          }
      }

      else if ((name.compareTo("`'msbs'") == 0) ||
(name.compareTo("`'lsbs'")==0)){
          int offs = (name.compareTo("`'msbs'")==0)?0:1;
          if (width > 1){
            Wire out = new Wire(env.getHierarchicalName()+"_o1",
width-1);
            if (Admin.debug){
                System.out.println("(VERILOG:) wire ["+(width-
2)+":0] "+out.getValue()+";");
            }
            try {
                Admin.addVariableDecl(out.getValue(), width - 1);
            } catch (BadVerilogException e){
                System.err.println(out.getValue()+" is an invalid
indentifier");
                throw new ApplyFailedException();
            } catch (DuplicationException e){
                System.err.println(out.getValue()+" is a
duplicate indentifier");
                throw new ApplyFailedException();
            }
            String instr = "assign "+out.getValue()+ " = ";
            if (width > 2){
                instr += "{ ";
                for (int i = width - 1; i>1; i--){
                  instr += ((Wire)A).getValue()+"["+(i-offs)+"],
";
                }
                instr += ((Wire)A).getValue()+"["+(1-offs)+"] };
";
            } else {
                instr += ((Wire)A).getValue()+"["+(1-offs)+"];";
            }
            if (Admin.debug){
                System.out.println("(Verilog:) "+instr);
            }
            Admin.addInstruction(instr);
            output = out;
          } else {
            BitVector out = new BitVector("'", 0);
            output = out;
          }
      }
```

```
      else if (name.compareTo("`'~'") == 0) {
          Wire out = new Wire(env.getHierarchicalName()+"_o1",
width);
          if (Admin.debug){
            System.out.println("(VERILOG:) wire ["+(width-1)+":0]
"+out.getValue()+";");
            System.out.println("(VERILOG:) assign "+
out.getValue()+
                           " = ~ "+((BitWire)A).getValue()+";");
          }
          try {
            Admin.addVariableDecl(out.getValue(), width);
          } catch (BadVerilogException e){
            System.err.println(out.getValue()+" is an invalid
indentifier");
            throw new ApplyFailedException();
          } catch (DuplicationException e){
            System.err.println(out.getValue()+" is a duplicate
indentifier");
            throw new ApplyFailedException();
          }
          Admin.addInstruction("assign "+ out.getValue()+
                         " = ~ "+((BitWire)A).getValue()+";");
          output = out;
      }

      else if ((name.compareTo("`'width'") == 0)){
          output = new Number(((BitWire)A).getWidth());
      }

      else {
          System.err.println("Unknown Un Vec operator");
          throw new ApplyFailedException();
      }

      try{
          env.setValue("X", output);
      } catch (DuplicationException e){
          System.err.println("Duplicate X");
          throw new ApplyFailedException();
      }
    }

    private Variable[] createPortsIn()
    {
      Variable[] inp = new Variable[1];
      inp[0] = new Variable("A");
      return inp;
    }
```

```
    private Variable[] createPortsOut()
    {
      Variable[] outp = new Variable[1];
      outp[0] = new Variable("X");
      return outp;
    }

}
```

A.2.27 STEntry.java

```
package LCSL;

/**
 * Symbol Table entry interface. All objects that can be entries
in
 * the symbol table must implement this
 *
 * @author Bogdan Caprita - bc2008@columbia.edu
 */

public interface STEntry
{
    public STEntry force(Environment env) throws
ForceFailedException;
    public boolean fullyEvaled();
}
```

A.2.28 Str.java

```
package LCSL;

/**
 * LCLS character string data type
 *
 * @author Bogdan Caprita - bc2008@columbia.edu
 */

public class Str implements Expression, STEntry
{
    String value;

    public Str(String value)
    {
      this.value = value;
    }

    public STEntry eval(Environment env)
```

```
    {
      return this;
    }


    public STEntry force(Environment env)
    {
      return this;
    }

    public String getValue()
    {
      return value;
    }

    public boolean fullyEvaled()
    {
      return true;
    }

    public void print(String offset)
    {
      System.out.println(offset+value);
    }
}
```

A.2.29 Sys.java

```
package LCSL;

/**
 * LCLS system.
 *
 * @author Bogdan Caprita - bc2008@columbia.edu
 */

public class Sys implements Expression
{
    private Expression component;

    public Sys(Expression component)
    {
      this.component = component;
    }

    public STEntry eval(Environment env) throws
EvalFailedException
    {
      STEntry comp = component.eval(env);
      try{
```

```
          comp = comp.force(env);
      }catch (ForceFailedException e){
          System.err.println("Could not resolve component name");
          throw new EvalFailedException();
      }

      if (comp instanceof CompPrim){
          Environment myEnv =
createSystemFrame(((CompPrim)comp).getPortsIn(),
((CompPrim)comp).getPortsOut());
          myEnv.setCompPrim((CompPrim)comp);
          if (comp instanceof Component){
            myEnv.setParent(((Component)comp).getEnv());
          }
          if (comp instanceof Primitive){
            /* this is needed only for printing out (debugging)
*/
            myEnv.setParent(env);
            if (((Primitive)comp).requiresForce){

Admin.remember(((Primitive)comp).needToBeForced(myEnv));
            }
          }
          return myEnv;
      }
      throw new EvalFailedException();
    }

    private Environment createSystemFrame(Variable[] portsIn,
Variable[] portsOut) throws EvalFailedException
    {
      Environment myEnv = new Environment(null);
      try{
          if (portsIn != null)
            for (int i = 0; i < portsIn.length; i++){
                myEnv.addEntry(portsIn[i].getName());
            }
          if (portsOut != null)
            for (int i = 0; i < portsOut.length; i++){
                myEnv.addEntry(portsOut[i].getName());
            }
      } catch(DuplicationException e){
          System.err.println("Duplication when declaring ports");
          throw new EvalFailedException();
      }
      return myEnv;
    }

    public void print(String offset)
    {
```

```
      System.out.println(offset+"System");
      component.print(offset+OFFSET);
    }
}
```

A.2.30 Thunk.java

```
package LCSL;

/**
 * Represents a thunk (encapsulation of name and environment of
evaluation
 * for the purpose of forcing)
 *
 * @author Bogdan Caprita - bc2008@columbia.edu
 */

public class Thunk implements STEntry
{
    private Variable exp;
    private Environment env;

    public Thunk(Variable exp, Environment env)
    {
      this.exp = exp;
      this.env = env;
    }

    public STEntry force(Environment env) throws
ForceFailedException
    {
      return exp.force(this.env);
    }

    public boolean fullyEvaled()
    {
      return false;
    }

    public String toString()
    {
      return "Thunk ["+ exp.getName()+","+
          env.getHierarchicalName()+"]";
    }

}
```

A.2.31 UndefinedException.java

```
package LCSL;

/**
 * Thrown when a lookup in the environment fails
 *
 * @author Bogdan Caprita - bc2008@columbia.edu
 */

public class UndefinedException extends Exception
{
}
```

A.2.32 Variable.java

```
package LCSL;

/**
 * Represents a name.
 * Contains most of the functionality needed to evaluate/force
names in LCLS
 * @author Bogdan Caprita - bc2008@columbia.edu
 */

public class Variable implements Expression, STEntry
{
    private String name;

    public Variable(String name)
    {
      this.name = name;
    }

    public STEntry eval(Environment env) throws
EvalFailedException
    {
      return new Thunk(this, env);
    }

    public STEntry force(Environment env) throws
ForceFailedException
    {
      if (env.markedForced(name)){
          System.err.println("Dependency loop detected!");
          throw new ForceFailedException();
      }

      STEntry entry;
      try{
          entry = env.lookupValue(name);
```

```
      }catch(UndefinedException e){
          System.err.println("Undefined name "+name+" in
"+env.getHierarchicalName());
          throw new ForceFailedException();
      }
      if (entry == null){
          env.force(env);
          try{
            env.markForced(name);
            STEntry forced_entry =
env.lookupValue(name).force(env);
            env.replaceValue(name, forced_entry);
            env.unmarkForced(name);
            return forced_entry;
          }catch(UndefinedException e){
            System.err.println("Undefined name "+name+" in
"+env.getHierarchicalName());
            throw new ForceFailedException();
          }
      }
      if (entry.fullyEvaled()){
          return entry;
      }
      if (entry instanceof Thunk){
          env.markForced(name);
          STEntry forced_entry = ((Thunk)entry).force(env);
          env.replaceValue(name, forced_entry);
          env.unmarkForced(name);
          return forced_entry;
      }
      throw new ForceFailedException();
    }

    public String getName()
    {
      return name;
    }

    public boolean fullyEvaled()
    {
      System.err.println("Tried to know if var "+name+" is fully
evaled");
      return false;
    }

    public void print(String offset)
    {
      System.out.println(offset+"Variable: "+name);
    }
}
```

A.2.33 Wire.java

```
package LCSL;

/**
 * Wire data type.
 * this data type is not available externally to LCLS
programmers. It
 * is an analogue of the Verilog wire and will be either
optimized away
 * by LCLS or will show up as a wire/input/output in the Verilog
file
 *
 * @author Bogdan Caprita - bc2008@columbia.edu
 */

public class Wire extends BitWire
{
    public Wire(String v, int w)
    {
      super(v, w);
    }
}
```

A.2.34 Body.java

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

/**
 * Container for a block of LCLS statements
 *
 * @author Bogdan Caprita - bc2008@columbia.edu
 */

public class Body implements Expression
{
    private Vector body;

    public Body(){
      body = new Vector();
    }

    public Body(Vector b) {
      body = b;
    }

    public STEntry eval(Environment env) throws
EvalFailedException
```

```
    {
      STEntry value = null;
      for (int i = 0; i < body.size(); i++){
          Expression ex = (Expression) body.elementAt(i);
          value = ex.eval(env);
      }
      return value; // last evaluated expression
    }

    public Vector getBodyVector() {
      return body;
    }

    public void addAll(Body b2) {
      Vector w = b2.getBodyVector();
      body.addAll(w);
    }

    public void add(Expression e) {
      body.add(e);
    }


    public void print(String offset)
    {
      System.out.println(offset+"{");
      for (int i = 0; i<body.size(); i++){
          Expression ex = (Expression) body.elementAt(i);
          ex.print(offset+OFFSET);
      }
      System.out.println(offset+"}");
    }

}
```

A.3 Driver, build files

A.3.1 Main.java

```
package LCSL;
import java.io.*;
import antlr.debug.misc.ASTFrame;
import antlr.CommonAST;

/***************
Main.java

@author Sachin Nene (srn28@columbia.edu)
```

```
The driver class for the LCSL language.
***********/

public class Main {

    public static void main(String[] args) {
      boolean debug = false;
      for (int i = 0; i < args.length; i++){
          if (args[i].compareTo("-debug") == 0){
            debug = true;
            Admin.debug = true;
          }
      }
      try {
          FileInputStream f = new FileInputStream(args[0]);
          LCSLLexer lexer = new LCSLLexer(f);
          LCSLParser parser = new LCSLParser(lexer);
          parser.program();
          f.close();

          CommonAST tree = (CommonAST)parser.getAST();
          if (tree != null) {

            //for debugging, we can print out the AST in JTree
            //ASTFrame frame = new ASTFrame("AST JTree Example",
tree);
            //frame.setVisible(true);

            LCSLWalker walker = new LCSLWalker();
            Body program = walker.body(tree);
            if (debug){
                program.print(" ");
                System.out.println("Environments before
eval:\n");
            }

            Environment global = Admin.createGlobalEnv();
            if (global == null){
                System.exit(1);
            }
            if (debug){
                global.print("");
            }
            try{
                program.eval(global);
            } catch (EvalFailedException e){
                if (debug){
                  e.printStackTrace();
                }
                System.out.println("Eval failed... Giving up!");
```

```
                System.exit(1);
            }
            if (debug){
                System.out.println("Environments after eval:\n");
                global.print("");
                System.out.println("Force Table:\n");
                Admin.printForceTable();
                System.out.println();
            }
            try{
                Admin.forceTable();
            } catch (ForceFailedException e){
                if (debug){
                  e.printStackTrace();
                }
                System.out.println("LCSL compilation failed. Go
buy Confluence!");
                System.exit(1);
            }
            if (debug){
                System.out.println("Environments after
force:\n");
                global.print("");
            }
          }

      }
      catch(IOException e) {
          System.err.println("IO exception occured");
          if (debug){
            e.printStackTrace();
          }
      }
      catch(antlr.RecognitionException e) {
          System.err.println("Recognition Error: "+e);
          if (debug){
            e.printStackTrace();
          }
      }
      catch(antlr.TokenStreamException e) {
          System.err.println("Token Stream Error: "+e);
          if (debug){
            e.printStackTrace();
          }
      }
      catch(Exception e) {
          System.err.println("Unexpected Error: "+e);
          if (debug){
            e.printStackTrace();
          }
```

```
      }

    }

}
```

A.3.2 build.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- Initial Set-Up -->

<project name="LCSL" default="main" basedir=".">

   <path id="lcsl.root">
     <pathelement location="${basedir}"/>
   </path>

 <!-- ============================================================
-->
 <!-- Initialization
-->
 <!-- ============================================================
-->
<!-- Setup project roots -->
 <target name="init">
    <property name="lcsl.root" value="${basedir}"/>
    <property name="lcsl.build" value="${lcsl.root}/build"/>
    <property name="lcsl.classes" value="${lcsl.build}/classes"/>
    <property name="lcsl.src" value="${lcsl.root}/src"/>
    <property name="lcsl.gen" value="${lcsl.build}/src"/>
    <tstamp>
      <format property="build.number" pattern="yyyyMMddHHmm"/>
    </tstamp>
</target>

<!-- Prepare src directory for compilation step -->

<target name="prepare-compile" depends="init">
    <mkdir dir="${lcsl.build}"/>
    <mkdir dir="${lcsl.classes}"/>
    <mkdir dir="${lcsl.gen}"/>
</target>

<target name="antlr" depends="prepare-compile">
    <antlr target="grammar.g" outputdirectory="${lcsl.gen}"/>
</target>

<target name="java" depends="antlr">
   <copy todir="${lcsl.gen}">
     <fileset dir="${lcsl.src}">
```

```
     </fileset>
   </copy>
   <javac srcdir="${lcsl.gen}" destdir="${lcsl.classes}">
   </javac>
</target>

<!-- Jar the application up -->
<target name="lcsl" depends="java">
   <jar jarfile="lcsl.jar" manifest="${lcsl.root}/resources/META-
INF/MANIFEST.MF">
     <fileset dir="${lcsl.classes}">
       <include name="LCSL/**"/>
     </fileset>
   </jar>
</target>


<target name="all" depends="lcsl">
</target>

<target name="clean" depends="init">
  <delete dir="${lcsl.build}"/>
</target>

<target name="main" depends="init">
  <echo message="-------------------------------------------------
------------"/>
  <echo message="welcome to LCSL build file"/>
  <echo message="Usage: ant [target-name]"/>
  <echo message="where [target-name] is one of the following
targets:"/>
  <echo message="antlr --> generates the antlr java files from
the grammar"/>
  <echo message="java --> compiles the antlr generated files"/>
  <echo message="all --> generates/compiles everything"/>
  <echo message="clean --> cleans up after itself removing the
build directory"/>
  <echo message="-------------------------------------------------
------------"/>
</target>

</project>
```

A.4 Testing suite

A.4.1 Test Files

```
::::::::::::::
CompInComp.lcsl
```

```
::::::::::::::
Xor <- comp +A +B -X
      X <- A '+' B
end

CompInComp <- comp +A +B +C -X
      Tmp <- {Xor, A B, _}
      Tmp1 <- {Xor, Tmp.X C, _}
      X <- Tmp1.X
end

Sys <- {CompInComp}

{VectorInput, "a" 1 2, Sys.A}
{VectorInput, "b" 2 2, Sys.B}
{VectorInput, "c" 3 2, Sys.C}
Out <- {VectorOutput}
Out.Name <- "x"
Out.Position <- 4
Out.Vector <- Sys.X

{Set, "FileName" "/home/chaue/parser/testing/LCSL-CompInComp.v",
_}
{Set, "BuildName" "CompInComp", _}
{Set, "GenTestBench" true, _}
{Set, "BenchName" "../../testing/CompInCompBench.v", _}
::::::::::::::
adder.lcsl
::::::::::::::
FullAdder <- comp +A +B +Cin -X -Cout
    X    <- A '^' B '^' Cin
    Cout <- (A '&' B) '|' ( B'&' Cin) '|' (Cin '&' A)
end

(* A N-bit ripple carry adder. *)
RippleAdder <- comp +A +B +Cin -X -Cout
  Eq <- {`==, ('width' A) 0, _}
  if Eq.X     (* Recursive termination condition. *)
    Cout <- Cin       (* Send the Cin right on through to Cout.
*)
    X    <- <>        (* Connect X to a zero width signal
constant. *)
  else
    (* Least significant bit full adder. *)
    {FullAdder, 'lsb' A 'lsb' B Cin, BitX BitCout}

    (* Recursive call to RippleAdder for remaining most
significant bits. *)
    {RippleAdder, 'msbs' A 'msbs' B BitCout, SubX Cout}
```

```
    (* Concatenation of results. *)
    {`'++', SubX BitX, X}
  end
end

Temp3 <- comp +A +B -X
  Tmp <- {RippleAdder, A B <0>, _ _}
  X <- Tmp@4
end

Sys <- {Temp3}

{VectorInput, "a" 1 2, Sys.A}
{VectorInput, "b" 2 2, Sys.B}
{VectorOutput, "x" 3 Sys.X, _}

{Set, "BuildName" "myadder", _}
{Set, "FileName" "../../testing/LCSL-adder.v", _}

::::::::::::::
allOps.lcsl
::::::::::::::
{Set, "FileName" "../../testing/LCSL-allOps.v", _}
{Set, "ModuleName" "allOps", _}

C <- 3
Ops <- comp +A +B -N -O -P -Q -R -S -T -U -V -W -X -Y -Z -AA -AB
      if C == 3
      N <- A '*' B
      O <- A '+' B
      P <- A '-' B
      Q <- A '&' B
      R <- A '^' B
      S <- A '|' B
      T <- A '<<' 3
      U <- B '>>' 3
      V <- A '<=' B
      W <- 'msbs' A
      X <- 'lsbs' B
      Y <- '~' A
      Z <- 'lsb' A
      AA <- 'msb' B
      AB <- A '+' '~'B
      end
end

Sys <- {Ops, _ _, _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _}

{VectorInput, "a" 11 4, Sys@1}
{VectorInput, "b" 2 4, Sys@2}
```

```
{VectorOutput, "n" 3 Sys@3, _}
{VectorOutput, "o" 4 Sys.O, _}
{VectorOutput, "p" 5 Sys.P, _}
{VectorOutput, "q" 6 Sys.Q, _}
{VectorOutput, "r" 7 Sys.R, _}
{VectorOutput, "s" 8 Sys.S, _}
{VectorOutput, "t" 9 Sys.T, _}
{VectorOutput, "u" 10 Sys.U, _}
{VectorOutput, "v" 11 Sys.V, _}
{VectorOutput, "w" 11 Sys.W, _}
{VectorOutput, "x" 11 Sys.X, _}
{VectorOutput, "z" 11 Sys.Z, _}
{VectorOutput, "aa" 11 Sys.AA, _}
{VectorOutput, "y" 11 Sys.Y, _}
{VectorOutput, "ab" 11 Sys.AB, _}
::::::::::::::
assignments.txt
::::::::::::::
(* assignments *)
A <- B
B <- {Foo, A B, $}
A <- 308944
A <- -349487 (*- causes error*
A <- 0x4587F45 (* F causes error*
A <- 0x45854f435 (*f causes error*
A <- 0b34134
A <- 0b34f34(*f causes error correctly*
A <- 0b01101110101
A <- 1b348932f (*f causes error correctly*
A <- 1b10101101
A <- 1x304897234
A <- 1x234fa34
A <- asdvedr (*causes error correctly*
A <- Aavkasdf
A<-4584
A <- Foo
$ <- true (* user defined var can be any char so long as it's not
a keyword*)
! <- 34
Q <- true
Q <- !false
Q <- maybe
Z <- A + 1
Z <- A || B
Z <- ! Sys.port
Z <- Sys.Port
Sys.port <- Sys@34
Sys.Port <- Sys@port
Z <- Sys@Pavd (*Pavd should be required to be an integer I think
*)
```

```
Z <- prim Target +A +B -C end
Z <- "string"
A <- "'string'"
A <- "$%@#$^%$+*#%^@|\/?<>,.adfl;a;vadsf"
Z <-
  comp +A +B -X
     X <- A + B
  end
(*Z <- '0111'  confluence accepts "Vector Constant" *)

(* assigning components *)

Sys <- {Foo, a b, x}
Sys <- {Foo, A B, X}
sys <- {foo, a b x, a}
Sys <- {foo, A B, X}
::::::::::::::
booleans.lcsl
::::::::::::::
T <- true
F <- false

Xor <- comp +A +B -X
      X <- A '^' B
end

Nor <- comp +A +B -X
      X <- '~' A '&' '~'B
end

Booleans <- comp +A +B +C -X -Y
      if !C
            Tmp <- {Xor, A B, _}
            X <- Tmp.X
      else
            Tmp <- {Nor, A B, _}
            X <- Tmp.X
      end
end

Sys <- {Booleans, _ _ T, _}

{VectorInput, "a" 1 4, Sys.A}
{VectorInput, "b" 2 4, Sys.B}
{VectorOutput, "x" 3 Sys.X, _}

{Set, "FileName" "/home/chaue/parser/testing/LCSL-booleans.v", _}
{Set, "BuildName" "booleans", _}

::::::::::::::
```

```
components.txt
::::::::::::::
(* components *)

component Foo +A +B -X
      X <- A ^ B
end
A <- f (*marker - causes error*)

component Bar +Q +G +H -Z -Cadfia -Casdklfjad343 (* chars like =-
(){}[]/\><'"; cause errors correctly  ; and " cause crazy errors
*)
      if Q
            Z <- {Foo, G H, $}
      else
            Z <- {Foo, {Bar, ! Q G H, $} H, $}
      end
end

component Xor +A +B -X
     A <- f (*marker - causes error *)
     if A < B
      X <- A ^ B
      P <- "stupid"
      Q<-34
     ef (A == B)then (*CAN WE GET RID OF THE NECESSARY THEN?? *)
     (*   X <-
        comp +A +B -X
           X <- A + B
        end *)  (*CANNOT DO THIS NOW! *)
     ef B != A then
     if B == "stupid"

     (* LISTS NOT IMPLEMENTED ef A == [] *)
     else
     end
     end (* no dangling else problem!*)

end


component Test +C +D +E -Y
      Y<-{Xor, {Xor, C D, $} E, $}
end

component Bar +G +H -Z
        Z <- f
      Z <- {Foo, G H, $}+{Foo, H G, $}
end
::::::::::::::
```

```
declarations.txt
::::::::::::::

(* declarations *)
:Foo
:Sys
:Xor

:Bar
:Sys3
:$
:!
:_adf
:A:
:A_d34d
::::::::::::::
decoder.lcsl
::::::::::::::

Decode <- comp +A +B +In -W -X -Y -Z
      W <- In '&' A '&' B
      X <- In '&' A '&' '~' B
      Y <- In '&' '~' A '&' B
      Z <- In '&' '~' A '&' '~' B
end

 Decoder <- {Decode, _ _ _, _}
{VectorInput, "a" 1 1, Decoder.A}
{VectorInput, "b" 2 1, Decoder.B}
{VectorInput, "in" 3 1, Decoder.In}
{VectorOutput, "w" 5 Decoder@4, _}
{VectorOutput, "x" 4 Decoder@5, _}
{VectorOutput, "y" 8 Decoder@6, _}
{VectorOutput, "z" 7 Decoder@7, _}

{Set, "BuildName" "decoder", _}
{Set, "FileName" "../../testing/LCSL-decoder.v", _}

::::::::::::::
infLoop.lcsl
::::::::::::::

Looper <- comp +A +B -X
      if 'width' A == 0
            X <- <>
      else
            Tmp <- {Looper, A B, _}
            X <- Tmp.X
      end
end
```

```
Sys <- {Looper}

{VectorInput, "a" 1 4, Sys.A}
{VectorInput, "b" 2 4, Sys.B}
{VectorOutput, "x" 3 Sys.X, _}

{Set, "BuildName" "infiniteloop", _}
{Set, "FileName" "/home/chaue/parser/testing/LCSL-infLoop.v", _}

{Set, "GenTestBench" true, _}
::::::::::::::
intIf.lcsl
::::::::::::::
{Set, "FileName" "../../testing/LCSL-intIf.v", _}
{Set, "BuildName" "allOps", _}


C <- 3
D <- 4
Ops <- comp +A +B +C -N -O -P -Q -R -S -T -U -V -W -X -Y -Z -AA -
AB
      if C == 3
      N <- A '*' B
      O <- A '+' B
      P <- A '-' B
      Q <- A '&' B
      R <- A '^' B
      S <- A '|' B
      T <- A '<<' 3
      U <- B '>>' 3
      V <- A '<=' B
      W <- 'msbs' A
      X <- 'lsbs' B
      Y <- '~' A
      Z <- 'lsb' A
      AA <- 'msb' B
      AB <- A '+' '~'B
      else
      N <- <>
      O <- <>
      P <- <>
      Q <- <>
      R <- <>
      S <- <>
      T <- <>
      U <- <>
      V <- <>
      W <- <>
      X <- <>
```

```
      Y <- <>
      Z <- <>
      AA <- <>
      AB <- <>
      end
end

Sys <- {Ops, _ _ C, _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _}

{VectorInput, "a" 11 4, Sys@1}
{VectorInput, "b" 2 4, Sys@2}
{VectorOutput, "n" 3 Sys.N, _}
{VectorOutput, "o" 4 Sys.O, _}
{VectorOutput, "p" 5 Sys.P, _}
{VectorOutput, "q" 6 Sys.Q, _}
{VectorOutput, "r" 7 Sys.R, _}
{VectorOutput, "s" 8 Sys.S, _}
{VectorOutput, "t" 9 Sys.T, _}
{VectorOutput, "u" 10 Sys.U, _}
{VectorOutput, "v" 11 Sys.V, _}
{VectorOutput, "w" 11 Sys.W, _}
{VectorOutput, "x" 11 Sys.X, _}
{VectorOutput, "z" 11 Sys.Z, _}
{VectorOutput, "aa" 11 Sys.AA, _}
{VectorOutput, "y" 11 Sys.Y, _}
{VectorOutput, "ab" 11 Sys.AB, _}


::::::::::::::
intOps.lcsl
::::::::::::::
A <- 1
B <- 2
C <- 3
D <- 4
E <- 5
F <- 6
G <- 7
H <- 8
I <- 9
J <- 0
K <- L
L <- K


IntOps <- comp +K +L -O
(*    if (K ** L) == K
            O <- <110>
      end *)
(*    if (K * L) == K  *)           (* L is 1 *)
```

```
      (*    O <- <000>
      else *)


      if (K + L) == 5               (* 1, 4 or 2, 3 *)
            O <- <001>
      else
      if K % L == 1                 (* K is 1 *)
            O <- <010>
      else
      if K / L == 0                 (* K is 0 *)
            O <- <011>
      else
      if K != L               (* K != L *)
            O <- <100>
      else
            O <- <101>        (* K == L *)
      end
      (* end *)
      end
      end
      end
end

Sys <- {IntOps, A K, _}

{VectorOutput, "o" 1 Sys.O, _}

{Set, "FileName" "/home/chaue/parser/testing/LCSL-intOps.v", _}
{Set, "BuildName" "intOps", _}
{Set, "GenTestBench" true, _}
{Set, "BenchName" "../../testing/noInputTestBench.v", _}
::::::::::::::
intloop.lcsl
::::::::::::::
Q <- 3

Loops <- comp +A +B +C +Q -D -E -F
      if Q == 0
            D <- A
            E <- B
            F <- C
      ef Q >=2
            G <- A '^' B
            H <- B '&' C
            I <- A '|' C
            Tmp <- {Loops, G H I (Q-1), _ _ _}
            D <- Tmp.D
            E <- Tmp.E
            F <- Tmp.F
```

```
      ef Q < 2
            G <- C '<<' 3
            H <- B '*' C
            I <- A '>>' 3
            Tmp <- {Loops, G H I (Q-1), _ _ _}
            D <- Tmp.D
            E <- Tmp.E
            F <- Tmp.F
      end
end

Sys <- {Loops, _ _ _ Q, _ _ _}

{VectorInput, "a" 3 4, Sys.A}
{VectorOutput, "d" 1 Sys.D, _}
{VectorInput, "b" 2 4, Sys.B}
{VectorInput, "c" 5 4, Sys.C}
{VectorOutput, "e" 6 Sys.E, _}
{VectorOutput, "f" 4 Sys.F, _}
(*
{VectorInput, "g" 9 4, Sys.G}
{VectorOutput, "h" 10 Sys.H, _}
*)

{Set, "FileName" "../../testing/LCSL-intloop.v", _}
{Set, "BuildName" "intloop", _}
::::::::::::::
multOut.lcsl
::::::::::::::
MultOut <- comp +A +B -C -D -E
      Tmp1 <- {`'^'}
      Tmp1.A <- A
      Tmp1.B <- B
      C <- Tmp1.X

      Tmp2 <- {`'+'}
      Tmp2.A <- A
      Tmp2.B <- B
      D <- Tmp2.X

      Tmp3 <- {`'|'}
      Tmp3.A <- A
      Tmp3.B <- B
      E <- Tmp3.X
end

Sys <- {MultOut}


{VectorOutput, "c" 3 Sys.C, _}
```

```
{VectorInput, "a" 1 4, Sys.A}
{VectorOutput, "d" 4 Sys.D, _}
{VectorInput, "b" 2 4, Sys.B}
{VectorOutput, "e" 5 Sys.E, _}

{Set, "FileName" "../../testing/LCSL-multOut.v", _}
{Set, "BuildName" "multout", _}

::::::::::::::
recursion.lcsl
::::::::::::::
Xor <- comp +A +B -X
      T <- {`'+', A B, _}
      X <- T.X
      (* T.A <- A
      T.B <- B
      X <- T.X *)
end


BusXor <- comp +A +B -X
      W <- {`'width'}
      W@1 <- A
      Eq <- {`==}
      Eq@1 <- W@2
      Eq@2 <- 0
      Result <- Eq@3
      if Result
            X <- <>
      else
            BitXor <- {Xor}
            Bit1 <- {`'lsb'}
            Bit1.A <- A
            Bit2 <- {`'lsb'}
            Bit2.A <- B
            BitXor.A <- Bit1.X
            BitXor.B <- Bit2.X
            Rest <- {BusXor}
            Bits1 <- {`'msbs'}
            Bits2 <- {`'msbs'}
            Bits1.A <- A
            Bits2.A <- B
            Rest.A <- Bits1.X
            Rest.B <- Bits2.X
            Concat <- {`'++'}
            Concat.A <- BitXor.X
            Concat.B <- Rest.X
            X <- Concat.X
      end
end
```

```
Sys <- {BusXor}

InA <- {VectorInput}
InB <- {VectorInput}
OutX <- {VectorOutput}

InA.Name <- "a"
InB.Name <- "b"
OutX.Name <- "x"

InA.Position <- 1
InB.Position <- 2
OutX.Position <- 3

InA.Width <- 2
InB.Width <- 2
Sys.A <- InA.Vector
Sys.B <- InB.Vector
OutX.Vector <- Sys.X

Set1 <- {Set}
Set1.Name <- "FileName"
Set1.Value <- "../../testing/outputrecursive.v"

Set2 <- {Set}
Set2.Name <- "ModuleName"
Set2.Value <- "RecursiveModule"

::::::::::::::
recursion2.lcsl
::::::::::::::
Xor <- comp +A +B -X
      X <- A '+' B
end

BusXor <- comp +A +B -X
      Width <- {`'width', A, _}
      Eq <- {`==, Width.X 0, _}
      if Eq.X
            X <- <1>
      else
            {`'lsb', B, SmallB}
            {`'msbs', A, RestA}
            {`'msbs', B, RestB}
            {BusXor, RestA RestB, Rest}
            {Xor, SmallA SmallB, BitX}
            {`'lsb', A, SmallA}
            {`'++', Rest BitX, X}
      end
```

```
end

Sys <- {BusXor, _ _, _}

  {VectorInput, "a" 1 3, Sys@1}
{VectorInput, "b" 2 3, Sys@2}
{VectorOutput, "x" 3 Sys@3, _}

{Set, "FileName" "/home/chaue/parser/testing/LCSL-recursion2.v",
_}
{Set, "BuildName" "test", _}
{Set, "GenTestBench" true, _}
{Set, "BenchName" "/home/chaue/parser/testing/recursionBench.v",
_}
::::::::::::::
test.txt
::::::::::::::

(* declarations *)
(*
:Foo
:Sys
:Xor

:Bar
:Sys3
*)
(* constant assignments *)
(*
A <- B
B <- {Foo, A B, $}
*)
(* A <- 308944
A <- -349487 (*- causes error*
A <- 0x4587F45 (* F causes error*
A <- 0x45854f435 (*f causes error*
A <- 0b34134
A <- 0b34f34(*f causes error correctly*
A <- 0b01101110101
A <- 1b348932f (*f causes error correctly*
A <- 1b10101101
A <- 1x304897234
A <- 1x234fa34
A <- asdvedr (*causes error correctly*
A <- Aavkasdf
A<-4584
A <- Foo
*)
$ <- true (* user defined var can be any char so long as it's not
a keyword*)
```

```
! <- 34
Q <- true
Q <- false
Q <- maybe
Z <- A + 1
Z <- A || B
Z <- Sys.port
Z <- Sys.Port
Sys.port <- Sys@34
Sys.Port <- Sys@port
Z <- Sys@Pavd (*Pavd should be required to be an integer I think
*)
(* Z <- '0111'  confluence accepts "Vector Constant" *)

(* components *)

component Foo +A +B -X
      X <- A ^ B
end
A <- f (*marker - causes error*)

component Bar +Q +G +H -Z -Cadfia -Casdklfjad343 (* chars like =-
(){}[]/\><'"; cause errors correctly  ; and " cause crazy errors
*)
      if Q
            Z <- {Foo, G H, $}
      else
            Z <- {Foo, {Bar, ! Q G H, $} H, $}
      end
end

component Xor +A +B -X
     A <- f (*marker - causes error *)
     if A < B
      X <- A ^ B
      P <- "stupid"
      Q<-34
     ef (A == B)then (*CAN WE GET RID OF THE NECESSARY THEN?? *)
     (*   X <-
        comp +A +B -X
           X <- A + B
        end *)  (*CANNOT DO THIS NOW! *)
     ef B != A then
     if B == "stupid"

     (* LISTS NOT IMPLEMENTED ef A == [] *)
     else
     end
     end (* no dangling else problem!*)
```

```
end


component Test +C +D +E -Y
      Y<-{Xor, {Xor, C D, $} E, $}
end

component Bar +G +H -Z
        Z <- f
      Z <- {Foo, G H, $}+{Foo, H G, $}
end


(* assigning components *)

(* Sys <- {Foo, a b, x}
Sys <- {Foo, A B, X}
sys <- {foo, a b x, a}
Sys <- {foo, A B, X}
*)
(* primitives *)
(* {VectorInput, "a" 1 4, Sys.A}
 {VectorInput, "b" 2 4, Sys.B}
 {VectorOutput, "x" 3   Sys.X}
  {VectorInput, "c" 4 4, Sys3.G}
 {VectorInput, "d" 5 4, Sys3.H}
{VectorInput, "e" 6 4, Sys3.C}
 {VectorOutput, "y" 7    Sys3.Z}

{Set, "FileName" "xor"}
{Set, "BuildName" "fireman"}
{Set, "GenVerilog" true}
{Set, "LevelDensity" 0}
{Set, "GenSignalMap" true}
*)
(* SOME PRIMITIVES DOESN'T REQUIRE OUTPUT *)
::::::::::::::
test1.txt
::::::::::::::
component SystemToString +A -X
  if !{IsSystem, A, Blah}
    Y <- {Error, "SystemToString:  Invalid type.  Expecting
system.", null}
     (*Y <- {Error, "A", null}*)
    X <- ""
  else
    X <- "<system>"
  end

end
```

```
::::::::::::::
test2.txt
::::::::::::::
(*X <- comp +A +Y -N V <- A  end*)
Q <- {R}
::::::::::::::
test3.txt
::::::::::::::
Dummy1 <- {VectorInput}
Dummy1.Name <- "foo"
Dummy1.Position<- 1
Dummy1.Width <- 4
Dummy2 <- {VectorOutput}
Dummy2.Name <- "bar"
Dummy2.Position <- 1
Dummy2.Vector <- Wire1
Wire1 <- Dummy1.Vector
::::::::::::::
test4.txt
::::::::::::::
A <- comp +B +C +E -D -F
  Z <- {Y}
  W <- V
  D <- B + (C - E * F)
  E <- !F

  K <- {Blah, 4 5 6 _, B}
  K@4 <- Blecht
end
::::::::::::::
test5.txt
::::::::::::::
B <- <10110101011>
C <- false
::::::::::::::
testOp.txt
::::::::::::::
A <- {`'+'}
```

A.4.2 Regression Testing Script

```
@files = qw/ adder allOps booleans CompInComp compPass decoder
intIf
intOps intloop multOut recursion recursion2 dependentComp /;

foreach $file (@files) {
    print $file . "\n";
    system "java -jar lcsl.jar testing/" . $file . ".lcsl";
    system "iverilog " . $file . "Bench.v -o " . $file ;
```

```
    system "./" . $file . " | diff - testing/results/" . $file .
"Bench.txt";
    unlink $file;
    unlink $file . "Bench.v";

}
```