Logisimple

Final Report

Yuanxia Lee (yl3262): yl3262@columbia.edu
Sarah Walker (scw2140): scw2140@columbia.edu
Kundan Guha (kg2632): kundan.guha@columbia.edu
Hannah Pierce-Hoffman (hrp2112): hrp2112@columbia.edu

December 20, 2017

Contents

1 Introduction

Logisimple allows the user to simulate circuits using a C-like syntax. Experienced programmers, perhaps with little hardware experience but a lot of software experience, can easily create combinational building blocks to quickly model complex hardware. The completed circuit can be treated as a "black box" during simulation, allowing users to manipulate inputs and view outputs for the entire system.

1.1 Motivation

Logisimple uses three simple building blocks: AND gates, OR gates, and inverters (NOT gates). These allow the user to build up bigger circuits based on the most simple boolean logic. Instead of complicated circuit diagrams and hardware, the user can simulate their circuits in software as if they were writing any other piece of code.

1.2 Features

- Built-in AND, OR, NOT
- Boolean data type
- C-like syntax
- User-defined gates
- I/O via command line interface

2 Language Tutorial

2.1 Environment Setup

Logisimple uses the following tools:
- OCaml, ocamlbuild, ocamlfind
- C, gcc
- BASH
- m4
- make

2.2 Compiler Setup

To compile Logisimple, type `make` from the `src` directory. Make sure to run make clean before compiling.

To compile a Logisimple program, run `./logisimple.native <FILENAME>.sim` to generate the LLVM IR.

To run a Logisimple program end-to-end, after compiling Logisimple, run the `compile_file.sh <LOGISIMPLE FILE> tester.c` script, which drives `tester.c`. Then run `./tester <INPUT 1> <INPUT 2> ... <INPUT N>` to run the compiled Logisimple file with inputs.

The printed output of the `tester` program is in the form: `<INPUT 1> <INPUT 2> ... <INPUT N> -> <OUTPUT>.`

2.3 Basic Example

Logisimple is a static language that's very similar to C. It includes type declarations and allows for custom types, of which all types consist of either booleans, boolean arrays, or gates for the simplicity of hardware description. A simple example program is shown below.

```
TICK(in1, in2){
    AND a(in1, in2);
    out = a;
}
```

The above program performs a bitwise `and` on two inputs. Note a toplevel function name. Here we call

it TICK, but the user may name it with any combination of letters that are all caps. This function naming with parameters will allow for ease of use for changing input values, since the program will be called like a function from a c wrapper.

3 Language Reference Manual

3.1 Introduction

Logisimple aims to provide a streamlined, text-based format for simulating digital circuits. With a structure based on nesting gates, users can combine sequential and combinational building blocks to quickly model complex hardware. The completed circuit can be treated as a "black box" during simulation, allowing users to manipulate inputs and view outputs for the entire system. Some of our most important goals in designing this language have been the following:

- Concise syntax
- Small number of types and operators
- Generally object-oriented concept of gates
- Intended for skilled users with knowledge of digital hardware

3.2 Lexical Elements

3.2.1 Identifiers

Gate names Both user-defined and built-in gate names are all uppercase letters. Examples:

```
AND x(0,0);
%% Declaring and assigning a new AND gate will always require AND in uppercase letters.
 ↪  The same applies for OR. %%

MYNEWGATE{
    % gate definition here
}
%% When defining a new gate type, always use uppercase names %%
```

Variables Boolean literals, boolean arrays, and instances of gates may be assigned to variables. Variables must begin with a lower case letter but can be followed by any combination of alphanumerics. Variables can also be declared without assignment and assigned later. Examples:

```
bool myVariableName;
bool[2] myArrayVariable2 = [0,0];
AND myGateVariable(0,1);
```

3.2.2 Keywords

out `out` is a keyword which is used to define the output of a user-defined gate. Example:

```
MYGATE{
    OR a(0,0);
    AND b(a,1);
    out = b;
}
```

Example when the output is an array that contains more than one element:

```
MYOTHERGATE(){
        out = [1, 0, 1];
}
```

include `include` is a keyword that is used to define external files to be included in the program. This facilitates the reuse of previously written code.

3.2.3 Operators

Indexing The bracket operator `[]` is used to index boolean arrays, which can either be from a gate with multiple outputs or a declared boolean array.

```
bool[5] barr = [0,1,1,0,1];
AND tmp(1,barr[1]);
% tmp evaluates to 1

MYGATE(){
    AND a(0,1);
    OR b(0,0);
    NOT c(b);
    out = [a, b, c];
}
MYGATE a;
bool b = a[2];
% b evaluates to 1
```

Assignment Our syntax supports the use of the = operator to set the outputs of a gate or perform another assignment to a variable name. We follow the standard convention for use of this operator: the value of the right-side operand is stored in the left-side operand.

Operator precedence In Logisimple, if there are multiple operators in a line, they are evaluated based on precedence. The operators listed below are in order of highest precedence first. Indexing evaluates left-to-right (left associative) whereas the assignment operator is evaluated right-to-left (right associative).

```
1. indexing []
2. assignment =
```

An example piece of code with an explanation of its evaluation:

```
bool[2] x = [1,0];
NOT x1(x[1]);
bool[2] y = [x1, x[0]];
%%The boolean array y is set to the output of x1 in the 0th indexed position, followed
↪  by the 0th index of boolean array x (which evaluates to 1) in the 1st indexed
↪  position.  %%
```

3.2.4 Separators

Commas The comma is used to separate multiple boolean inputs or outputs.

```
bool[3] x = [0,0,0];
MYGATE y (a,b) {
    out = [a,b];
}
```

Semicolons Semicolons are used as statement terminators. Any line that is not a bracket must end in a semicolon.

```
IDEN(a) {        # no semicolon needed here
    out = a;     # semicolon needed here
}
```

Comments The percent symbol is used to denote comments.

```
% Single line comments begin with a single pound symbol.
%% Multi line comments are enclosed by two pound symbols
on either side. %%
```

Brackets Logisimple employs two types of brackets: `[]` (square brackets), and `{}` (curly brackets). Square brackets are used to enclose boolean arrays and index arrays. User-defined gate definitions use curly brackets to delimit the beginning and end of the definition.

```
   TICK(a,b,c){
        % some source code
        bool[2] myarr = [1,0]; # boolean array is enclosed in square brackets
    }
```

Brackets also control the scope of identifiers. See Section 4 for details.

3.2.5 Whitespace

Alphanumerics must be separated by some non-alphanumeric (whitespace, commas, parantheses, etc).

3.3 Data types

3.3.1 Booleans

Booleans are defined as `bool id = e`, where `e` can be either 1 or 0, representing the usual binary notion of the symbols. `e` can also be any valid expression that evaluates to a boolean value– for instance, `e` could be another boolean variable, or the output of a gate.

3.3.2 Boolean arrays

Arrays of boolean values are declared as `bool[n] id = [e1, e2, ...]`, where `id` is any identifier for the variable, `e1` are any valid expressions evaluating to booleans, and `n` is the number of inputs.

```
% Example 1:
bool a = 1;
bool[2] one = [a,0];

% Example 2:
AND x(1,0);
bool[2] two = [x, 1];
```

3.3.3 User-defined combinational logic gates

Gates are the foundational data type, out of which all simulation elements are constructed. Gates can be very large and complex, comprising an entire circuit with many subgates, or very small, performing a single basic operation. The two essential components of a user-defined gate definition are the *out* statement, which consists of one or more outgoing pins with boolean values, and the *body*, which describes how the output is calculated. The input values are set when instances of a user -defined gate are actually declared. The body of a gate may contain subgates that operate on the gate inputs; these subgates may be previously defined or defined within the body of the supergate. We provide AND, OR, NOT, and DFF primitive gate types, as well as other common gate types in our standard library.

Output The last line of a user-defined gate definition must be the output. The output line specifies one or more outputs. **This line is mandatory in user-defined gates.** User-defined gates may output constant

values, boolean variables, or a combination of both.

```
MYGATE {
    % gate definition begins here
    % the variable x is defined somewhere
    out = x;
}
```

Body Before the output line of a user-defined gate definition, the body section allows the user to create variables, declare primitive gates, and define subgates. The user can also reference any gates or variables defined one scope level above the gate currently being defined (i.e., when defining a gate within one set of brackets, the definition may reference any global gates or variables). For simple user-defined gates, it is possible that the gate definition body may be empty.

```
% User-defined NOR gate
MYNOR(a,b){
        OR x(a,b);  % body
        NOT x1(x);
        out = x1;       % output
    }
```

3.3.4 Primitive combinational logic gates

AND AND is built in to Logisimple. An AND gate can be instantiated as follows:

```
AND mygate(1,1);
```

The above line instantiates an AND gate called mygate, and gives inputs 1 and 1 to the AND gate. Calling `mygate` would give the result of the logical AND of 1,1 which equals 1.

OR Similarly to AND, OR is a primitive in Logisimple. It is instantiated the same way as an AND gate, except with the word `OR` instead.

3.3.5 User-defined sequential logic gates

Flip Flops While in reality flip flops can be represented as a combination of logic gates, in Logisimple flip flops are represented through a base primitive type called `DFF` (D Flip Flop) and a global clock. All other flip flop types may be constructed from a combination of D Flip Flops and logic gates.

`DFF` models a normal one input, one output D Flip Flop where the input updates the stored value, and the output is the currently stored value. All D Flip Flops initially store 0. Flip flops operate on a shared global clock. Within each clock step they simply output based on the last clock step's inputs. Any given inputs update the stored value for the next step. For example,

```
DFF d1(1); %Currently stores 0, next step will store 1
AND a(d1,1); %On first cycle will output 0, second cycle 1
```

Clock All programs operate on a global clock driving all flip flops at once. See 3.5.1.

3.4 Scoping

An identifier's scope is the curly brackets which enclose it, and any brackets those brackets enclose, for any level of depth. Parameters which become populated via the c wrapper are defined within the entire source program because they are only enclosed in the outermost parentheses outside the outermost brackets.

Example:

```
TICK(a,b,c){
    AND x(a,b);
    MYCIRCUIT{
        OR y(a,c);
        NOT z(y);
        out = z;
    }
}
```

In the above example, a, b, c and x are declared throughout the entire source program. y and z are only defined within MYCIRCUIT. Note that a, b, and c are also defined in MYCIRCUIT.

3.5 Standard Library

The Standard Library provides commonly used gates in addition to the existing AND and OR primitives. These are available in the stdlib directory.

3.5.1 NAND

```
%% NAND function, implemented using AND and NOT
Author: Sarah Walker

Inputs: stdlib-in1, stdlib-in2
Outputs: stdlib-nand %%

AND stdlib-nand-and(stdlib-in1, stdlib-in2);
NOT stdlib-nand(stdlib-nand-and);

out = stdlib-nand;
```

3.5.2 NOR

```
%% NOR function, implemented using NOT and OR
Author: Sarah Walker

Inputs: stdlib-in1, stdlib-in2
Outputs: stdlib-nor %%

OR stdlib-nor-or(stdlib-in1, stdlib-in2);
NOT stdlib-nor(stdlib-nor-or);

out = stdlib-nor;
```

3.5.3 XOR

```
%% XOR function, implemented using NAND and OR
Author: Sarah Walker

Inputs: stdlib-in1, stdlib-in2
Outputs: stdlib-xor-and %%

NAND stdlib-xor-nand(stdlib-in1, stdlib-in2);
OR stdlib-xor-or(stdlib-in1, stdlib-in2);
AND stdlib-xor-and(stdlib-xor-nand, stdlib-xor-or);

out = stdlib-xor-and;
```

3.5.4 XNOR

```
%% XNOR function, implemented using NOT and XOR
Author: Sarah Walker

Inputs: stdlib-in1, stdlib-in2
Outputs: stdlib-xnor %%

XOR stdlib-xnor-xor(stdlib-in1, stdlib-in2);
NOT stdlib-xnor(stdlib-xnor-xor);

out = stdlib-xnor;
```

3.5.5 MUX-TWO

```
%% Two-input, one-output multiplexer, implemented using NOT, AND, and OR

Author: Hannah Pierce-Hoffman

Inputs: stdlib-in0, stdlib-in1, stdlib-a
Outputs: stdlib-mux-two %%

NOT stdlib-not-a(stdlib-a);

AND stdlib-top-and(stdlib-not-a, stdlib-in0);
AND stdlib-bottom-and(stdlib-a, stdlib-in1);

OR stdlib-mux-two(stdlib-top-and, stdlib-bottom-and);

out = stdlib-mux-two;
```

3.5.6 MUX-FOUR

```
%% Four-input, one-output multiplexer, implemented using NOT, AND, and OR

Author: Hannah Pierce-Hoffman

Inputs: stdlib-in0, stdlib-in1, stdlib-in2, stdlib-in3, stdlib-a, stdlib-b
Outputs: stdlib-mux-four %%

NOT stdlib-not-a(stdlib-a);
NOT stdlib-not-b(stdlib-b);

AND stdlib-top-and0(stdlib-not-a, stdlib-not-b);
AND stdlib-bottom-and0(stdlib-top-and0, stdlib-in0);

AND stdlib-top-and1(stdlib-a, stdlib-not-b);
AND stdlib-bottom-and1(stdlib-top-and1, stdlib-in1);

AND stdlib-top-and2(stdlib-not-a, stdlib-b);
AND stdlib-bottom-and2(stdlib-top-and2, stdlib-in2);

AND stdlib-top-and3(stdlib-a, stdlib-b);
AND stdlib-bottom-and3(stdlib-top-and3, stdlib-in3);

OR stdlib-top-or(stdlib-bottom-and0, stdlib-bottom-and1);
OR stdlib-bottom-or(stdlib-bottom-and2, stdlib-bottom-and3);

OR stdlib-mux-four(stdlib-top-or, stdlib-bottom-or);

out = stdlib-mux-four;
```

3.5.7 1-2-DEC

```
%% 1:2 Decoder, implemented using NOT
Author: Sarah Walker

Inputs: stdlib-in1
Outputs: stdlib-in1, stdlib-not-literal %%

NOT stdlib-not-literal(stdlib-in1)

out = [stdlib-in1, stdlib-not-literal];
```

3.5.8 2-4-DEC

```
%% 2:4 Decoder, implemented using NOT and AND
Author: Sarah Walker

Inputs: stdlib-in1, stdlib-in2
Outputs: stdlib-dec1, stdlib-dec2, stdlib-dec3, stdlib-dec4 %%

NOT stdlib-not-in1(stdlib-in1);
NOT stdlib-not-in2(stdlib-in2);

AND stdlib-dec1(stdlib-in1, stdlib-in2);
AND stdlib-dec2(stdlib-in2, stdlib-not-in2);
AND stdlib-dec3(stdlib-not-in1, stdlib-in2);
AND stdlib-dec4(stdlib-not-in1, stdlib-not-in2);

out=[stdlib-dec1, stdlib-dec2, stdlib-dec3, stdlib-dec4];
```

3.6 Program structure

3.6.1 File format and extension

All source programs must be enclosed in curly braces to denote the start and end of the source code, following TICK(e1, e2,...,en), where e1 through en are some variable names that represent boolean values. The file containing the source code requires file extension `.sim`.

3.6.2 Executable file

The compiler builds an executable file which simulates the behavior of the circuit. This executable file is `<filename>.simx`. The executable file must be run in a c program that will call TICK to allow boolean inputs to the program.

3.6.3 Using the standard library

The `include` keyword makes it simple for the user to access the standard library, as we are using M4 directives. Including an external file containing the standard library code effectively copies and pastes the standard library code into the current file.

3.7 Sample program

```
% Implementing a Half-Adder in Logisimple
TICK(in1,in2){
% These gates are implemented in the standard library, but shown here as an example

NAND(c,d) {
    AND a(c,d);
    NOT b(a);
    out = b;
}

XOR(c,d) {
    OR o(c,d);
    NAND na(c,d);
    AND a(o,na);
    out = a;
}

AND a(in1, in2);
XOR x(in1, in2);

bool carry = a;
bool result = x;

out = result, carry;
}
```

4 Project Plan

4.1 Planning Process

When beginning planning, we started by deciding on the type of language we wanted and subsequently divvying up the roles among our four team members based off abilities and desires. We then established a weekly meeting time that also functioned as a weekly deliverable time, in which deliverables were set accordingly based off the class' proposal, LRM, and "hello world" deadlines. Unfortunately, once we reached the "Hello World" program, our team fell behind as we were mired in trying to understand codegen. This pushed back all our deliverables with "Hello World" becoming our main priority for a few weeks. Once we moved past this issue, we returned to specific goals (albeit on a much tighter schedule) to complete the project in time.

4.2 Specification Process

When specifiying our language, we wanted it to be unique but made hardware description easy for the user. Unfortunately, in trying to be unique, some (read: many) of our ideas were consequently more difficult for the user, more difficult for us, and syntactic sugar. This resulted in a trimming process to slim down the language into its current minimal version that's very reminiscent of C. This ultimately also aided the codegen process, and we like to think minimalist design is user-friendly.

4.3 Development Process

Ideally, our development would have been a linear ascension from solidifed grammar to working compiler, but it was more akin to a bowline knot. Instead of building from a fleshed out scanner and parser to generating code and features, we constructed ideas and programs of what our language syntax, scanner, parser, and

AST would look like that was ultimately scrapped when we tried to implement with codegen. Thus, we implemented the simplest Hello World program possible with a minimal parser and code generator that was then fleshed out with a fuller AST and netlist that would serve as the bridge between front-end and back-end. After which, front-end and back-end could each become more fully formed.

4.4 Programming Style Guide

Though there were no strict conventions, we followed the following guidelines in our code:
- OCaml's formatting style for our compiler
- C-like style for Logisimple language (indentations, braces)
- Logisimple languages stored in files with .sim extension
- Gate declarations are in all caps; variables begin with lowercase followed by any combination of alphanumerics that is not a keyword

Look to the Language Reference Manual for more specifics, and so long as the code follows the rules of the language, particular style is up to the user (though it is recommended to follow our examples).

4.5 Project Timeline

4.6 Team Roles

4.6.1 Yuanxia Lee: Language Guru

As the Language Guru, I was responsible for understanding the ins-and-outs of the language, ie. ensuring the correctness of our LRM, last say in syntax style, etc. Otherwise, I assisted Kundan in ensuring the front-end worked by fixing errors across the board, testing the parser, and printing the AST.

4.6.2 Sarah Walker: Tester

My official role was "tester," so I designed and wrote the test suite. I had test-to-pass and test-to-fail categories, with the test-to-pass tests slightly more extensive. We needed to test our compiler's ability to handle the promised features and syntax as well as its ability to compute logical ANDs, ORs, NOTs, as well as all operations implemented in the standard library.

Outside of my role as tester, I often pair-programmed with Hannah while we worked on the initial Hello World which parsed ANDs and ORs with hardcoded variable inputs (before we had the ability to take inputs via wrapper function). Hannah and I designed the original codegen.ml to generate LLVM's logical AND, OR, and NOTs.

4.6.3 Kundan Guha: System Architect

My job was to create a functional front-end (scanner, parser, semant, AST), and as System Architect, guide the construction of the language pipeline. Once the pipeline was created, I helped in fleshing out individual programs of code such as codegen, to handle more than "hello world."

4.6.4 Hannah Pierce-Hoffman: Manager

My most significant role was to oversee the operations of the team. I was responsible for scheduling and leading meetings, checking in with each team member to assess the state of work being done at any given time, and stressing out constantly about deadlines. One of my most important logistical accomplishments was initiating the "Turning Point Meeting," in which our team met with Professor Edwards for 4 hours without a break and finally gained the knowledge necessary to understand Hello World.

Outside of my manager role, I worked a lot on the initial concept for Logisimple and the earlier drafts of the LRM. I also worked closely with Sarah on the subproject of implementing the original Hello World, which included a stripped-down front-end and the first working version of codegen.ml.

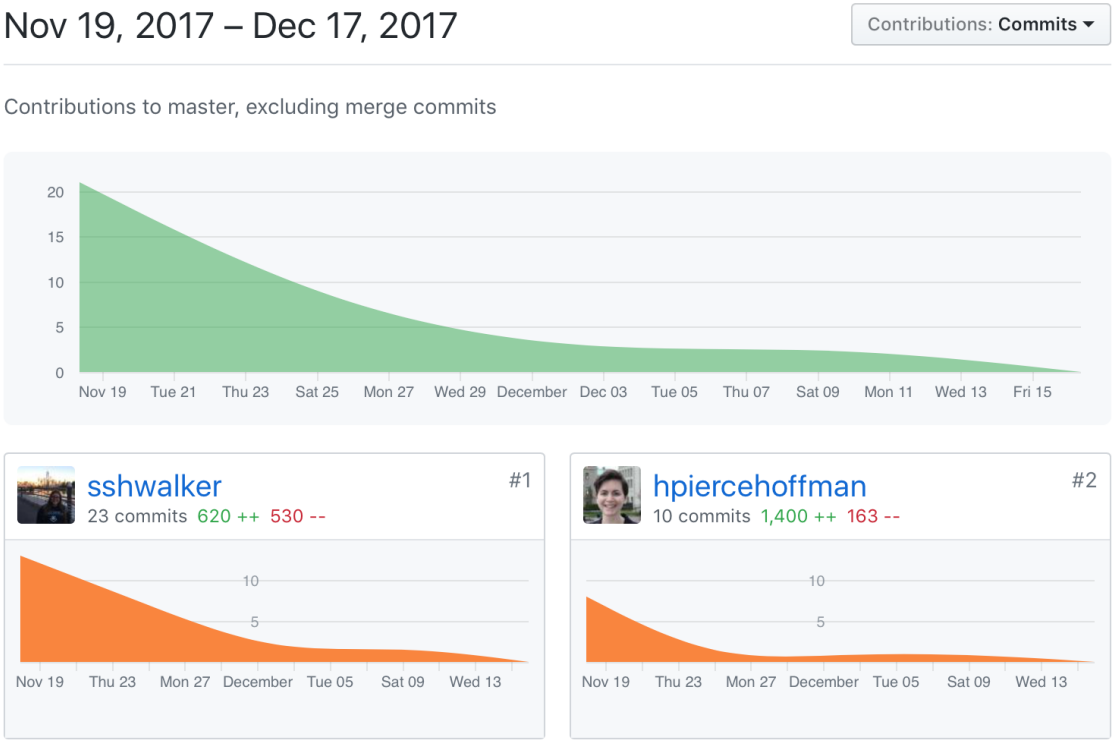
4.7 Software Development Environment

Operating Systems: Mac OS Systems, Ubuntu 14.04
Languages: OCaml (install with OPAM), C (for the wrapper)
Program Generators: Ocamlyacc, Ocamllex
Text Editor: Vim, Sublime
Version Control: Github

4.8 Project Log

4.8.1 Hannah and Sarah's Hello World repository

A repository created just for a Hello World, almost from scratch, over Thanksgiving Break. Here are the **contributions graphs:**



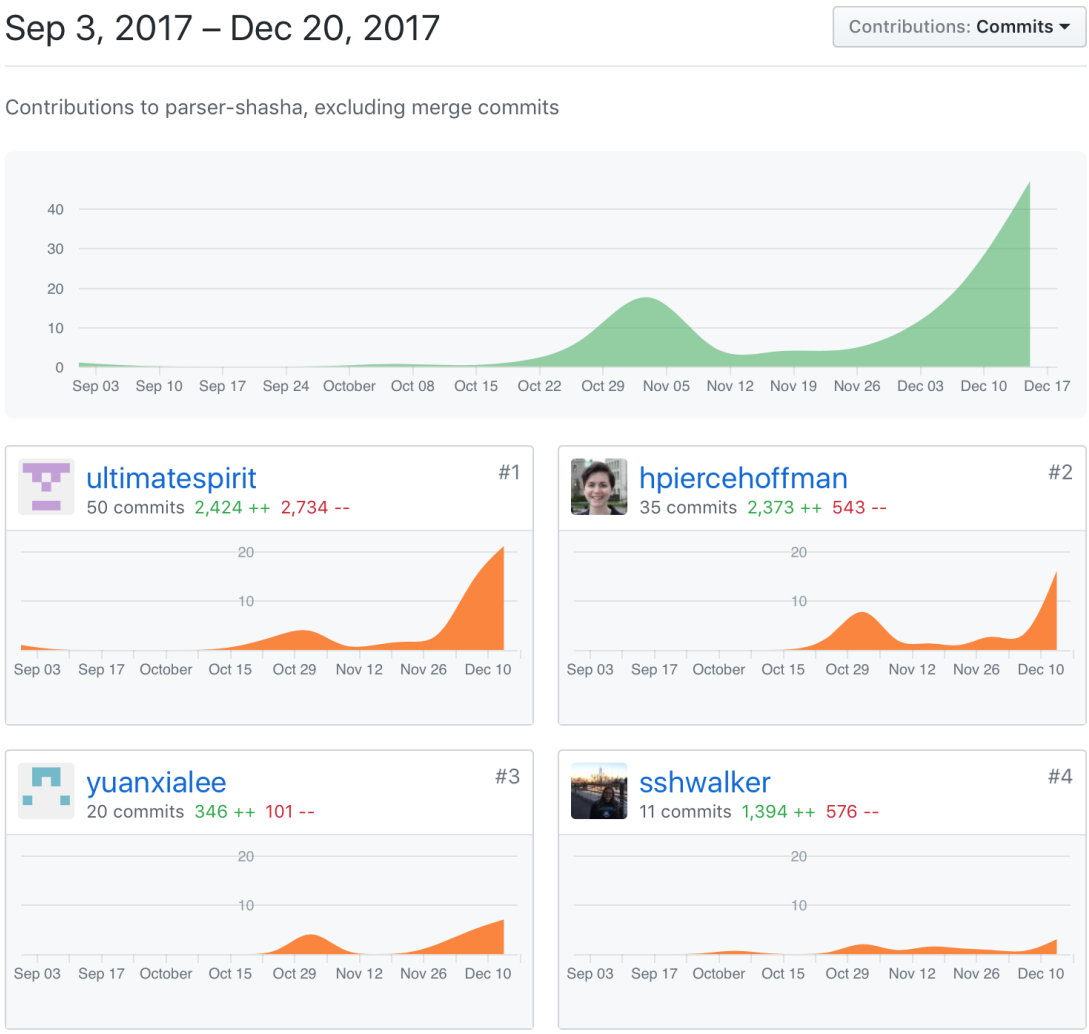
Hannah and Sarah Hello World log:

b606dee - Sarah Walker, 3 days ago : [StringMap] Trying to get variables and statements as separate lists in AST
419f49d - Sarah Walker, 3 days ago : why
ec67a13 - Hannah Pierce-Hoffman, 3 days ago : Merge branch 'master' of https://github.com/hpiercehoffman/plit
65244f9 - Hannah Pierce-Hoffman, 3 days ago : Integ directory in case it ever works
f6c1f88 - Sarah Walker, 4 days ago : [fix] restoring to working order
7e25189 - Sarah Walker, 12 days ago : [testing] Cleaned up testing directory and added print statements to testall script
abddce7 - Hannah Pierce-Hoffman, 12 days ago : builder multiple statements WORKINGls! made various changes to accomplish this, also renamed test files for ease of typing their names while testing. Still needs work passing outputs to the next prim, time for stringmap
99d51ca - Sarah Walker, 3 weeks ago : [multiple statements] two statement tests passing
056c270 - Sarah Walker, 3 weeks ago : [multiple statements] Added test case for multiple statement
3683ee0 - Sarah Walker, 3 weeks ago : [multiple statements] Non-working version of adding multiple statements. Problem with builder.
ba11b96 - Sarah Walker, 3 weeks ago : [clean up] Renamed ast.ml to netlist.ml

c8f379d - Sarah Walker, 3 weeks ago : [testing] Began writing tests, added test script
0e90b32 - Sarah Walker, 3 weeks ago : [NOT] CHECK OUT THAT NOT, but also really check out that llvm IR, it's clever with the XOR
8d2fbd7 - Sarah Walker, 3 weeks ago : [OR gates] Added functionality for OR gate; still only parses single statement at a time. See current hello.sim for what can now also be accepted.
c82190d - Sarah Walker, 3 weeks ago : Added useful info to README
b279163 - Sarah Walker, 3 weeks ago : RETURNING THE RESULT OF AN AND
56caf5f - Sarah Walker, 3 weeks ago : LOADED THOSE VARIABLES BACK AND AND'D THEM
212ea85 - Sarah Walker, 3 weeks ago : WE ALLOCATED A VARIABLE
6c9d307 - Hannah Pierce-Hoffman, 3 weeks ago : RUNNING A PROGRAM, IT RUNS, IT RUNS
a334af6 - Hannah Pierce-Hoffman, 3 weeks ago : so i have stripped EVERYTHING in the codegen out to only make a small function called main, fully ignoring the stringmap and all that stuff, thus producing a non functional IR, so this is not the most useful thing in the world, but the point is, IT COMPILES
4c764a1 - Sarah Walker, 3 weeks ago : Added a StringMap for identifiers before the expr function. Treating prims as functions for the purposes of lookups of output params. Having a weird problem with the expr matching; getting type errors. We rewrote everything again, including the AST to have a layer called stmt which allows IDs and Prims to be matched at the same time see lines 36-45 in codegen.
db28e1f - Sarah Walker, 3 weeks ago : Very close. Needs StringMap for identifiers.
3a36aa4 - Hannah Pierce-Hoffman, 3 weeks ago : it is almost building, have fixed many compiler errors but something is not quite right in codegen
284ca38 - Hannah Pierce-Hoffman, 3 weeks ago : it has reached the stage where compiler now produces useful messages
4d78b87 - Hannah Pierce-Hoffman, 3 weeks ago : parsing single andgate without main enclosure, various updates
7931dfc - Hannah Pierce-Hoffman, 3 weeks ago : just moving some code around
a357a31 - Sarah Walker, 4 weeks ago : goodnight ocaml you win
66e12d1 - Sarah Walker, 4 weeks ago : Moved my codegen to codegensarah.ml. Added Makefile for my codegen. Unclear why it won't actually compile.
585bd51 - Sarah Walker, 4 weeks ago : Starting work on AST
1695852 - Sarah Walker, 4 weeks ago : trying to fix my git screw up
fa14b46 - Sarah Walker, 4 weeks ago : Changed parser to ONLY handle the contents of edwards-help/hello.sim, which I also changed
a4cae31 - Sarah Walker, 4 weeks ago : Started on the backend
79f11e3 - Sarah Walker, 4 weeks ago : Added help files from Edwards
fa2f7af - Hannah Pierce-Hoffman, 4 weeks ago : starter pack
b60526e - Hannah Pierce-Hoffman, 4 weeks ago : Initial commit

After we achieved this version of Hello World, we integrated our changes with Kundan's PLT-2017Fall repository and we all worked from that repository instead.

4.8.2 Full Team Repository



PLT-2017Fall Repository (all members):

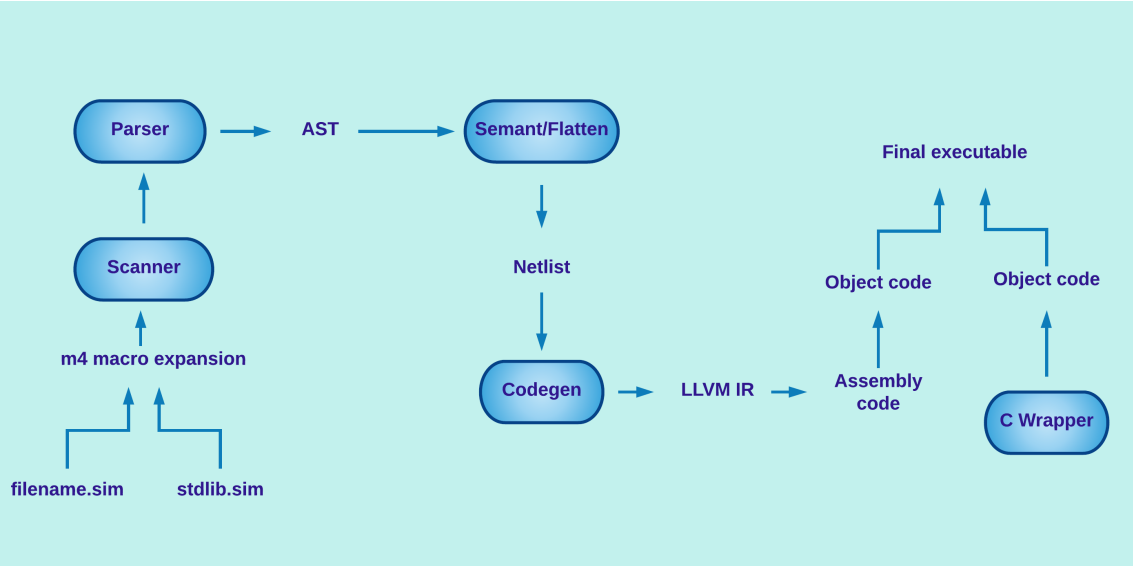
9dbf5c4 - Kundan Guha, 2 hours ago : Fixed not error
7ddf57b - Kundan Guha, 2 hours ago : Fix other error
382c477 - Kundan Guha, 2 hours ago : Fix error introduced in last commit
906af0c - Kundan Guha, 2 hours ago : Changed codegen base type to 1bit
7fb50e7 - Hannah Pierce-Hoffman, 3 hours ago : NOT in flatten
cb47e82 - Hannah Pierce-Hoffman, 3 hours ago : fixed trailing comma in tester
b416955 - Hannah Pierce-Hoffman, 3 hours ago : multi inp tester
0fd27e3 - Hannah Pierce-Hoffman, 3 hours ago : improved tester
6899f08 - Kundan Guha, 3 hours ago : updated demo to have real semant
8c510d7 - Kundan Guha, 4 hours ago : Semant is potentially finished
fcb8552 - Hannah Pierce-Hoffman, 4 hours ago : added tester file
2ef3b3c - Hannah Pierce-Hoffman, 4 hours ago : Merge branch 'parser-shasha' of https://github.com/ultimatespirit/PLT-2017Fall into parser-shasha v27ec70b - Hannah Pierce-Hoffman, 4 hours ago : First demo with the full pipeline is complete! Run ./compile_file.sh and _full.sim tester.c to see it in action
18209ab - Sarah Walker, 5 hours ago : Merge branch 'parser-shasha' of https://github.com/ultimatespirit/PLT-2017Fall into parser-shasha
966ec9d - Sarah Walker, 5 hours ago : [tests] Added pass and fail folders for tests. Added tests for include. Updated syntax on all tests to include out keyword

e3dd2ed - Hannah Pierce-Hoffman, 5 hours ago : Merge branch 'parser-shasha' of https://github.com/ultimatespirit/PLT-2017Fall into parser-shasha
d87ed1f - Hannah Pierce-Hoffman, 5 hours ago : made new demo directory with bridge function and trying to make it work
3110a32 - Kundan Guha, 5 hours ago : Fixed tiny error in netlist where program was in fact not a program
79db28c - Kundan Guha, 7 hours ago : Merge branch 'parser-shasha' of github.com:ultimatespirit/PLT-2017Fall into parser-shasha
d868d4e - Kundan Guha, 7 hours ago : Small changes to flatten, started makign expr conversion routine.
42bc625 - Hannah Pierce-Hoffman, 7 hours ago : list.concat added
f5cda9f - Yuanxia Lee, 19 hours ago : Commented simple semant with instructions
06f615b - Yuanxia Lee, 20 hours ago : Slight cleanup to flatten
86475f3 - Hannah Pierce-Hoffman, 26 hours ago : Merge branch 'parser-shasha' of https://github.com/ultimatespirit/PLT-2017Fall into parser-shasha
d158e0e - Hannah Pierce-Hoffman, 26 hours ago : first draft of flatten.ml complete except no consideration for A.Assigns, but A.Pdec and A.Gdec considered, should output finalized list of N.Expr that is theoretically ready for codegen
8cde9d0 - Kundan Guha, 2 days ago : Semant mostly finished, still has syntax errors to work out
6c896a7 - Hannah Pierce-Hoffman, 2 days ago : fixed two syntactic issues in flatten, added op conversion, and more clearly specified the Netlist item being constructed from A.Prim
a604186 - Hannah Pierce-Hoffman, 2 days ago : Started flatten.ml with case to handle Pdec; see comments for detailed explanation of operation of flatten and plan for remainder of code
0d435bf - Hannah Pierce-Hoffman, 2 days ago : fix typo
d391dbf - Yuanxia Lee, 2 days ago : logisimple.ml merge conflict resolved hopefully
79b09c5 - Yuanxia Lee, 2 days ago : Merge branch 'parser-shasha' of github.com:ultimatespirit/PLT-2017Fall into parser-shasha
0ff4cf7 - Yuanxia Lee, 2 days ago : Begin flatten, correct netlist comments
fddf345 - Kundan Guha, 2 days ago : Update on semant, still not done.
1b43e2e - Yuanxia Lee, 3 days ago : Finished AST printing
5f9990b - Kundan Guha, 2 days ago : Fixed small errors introduced in last commit to ast
1d14877 - Kundan Guha, 2 days ago : Merge branch 'parser-shasha' of github.com:ultimatespirit/PLT-2017Fall into parser-shasha
8481f63 - Kundan Guha, 2 days ago : Preliminary semant, modified ast and parser for array indexing.
e15984d - Yuanxia Lee, 3 days ago : AST printing testing
b7d6f3c - Kundan Guha, 3 days ago : Did in fact not add includes. Have them now.
5e95068 - Kundan Guha, 3 days ago : Maybe added include statements to scanner (Hannah + Me)
95089b2 - Kundan Guha, 3 days ago : Merge branch 'master' into parser-shasha
f46c4eb - Hannah Pierce-Hoffman, 3 days ago : array indexing to parser and AST
a03a62c - Kundan Guha, 3 days ago : Changed comment token to
62242b2 - Kundan Guha, 3 days ago : Merge branch 'master' of github.com:ultimatespirit/PLT-2017Fall
ba0229d - Kundan Guha, 3 days ago : Two outputs work
ccbdc01 - Hannah Pierce-Hoffman, 3 days ago : Merge branch 'master' of https://github.com/ultimatespirit/PLT-2017Fall
1983f31 - Hannah Pierce-Hoffman, 3 days ago : fix thing
dcf02dd - Sarah Walker, 3 days ago : Merge branch 'master' of https://github.com/ultimatespirit/PLT-2017Fall
69d7e74 - Sarah Walker, 3 days ago : [build] tester now prints instead of requiring echo 0
ddd04f3 - Kundan Guha, 3 days ago : Cleaned up cleanup directory
17568b1 - Yuanxia Lee, 3 days ago : Updated compile file bash script
4b05a01 - Kundan Guha, 3 days ago : Compilation shells cript, for some reason llc does not like being called within script.
f2afa72 - Sarah Walker, 3 days ago : [build] Changed Custom to TICK in tester.c
e60b975 - Hannah Pierce-Hoffman, 3 days ago : fixed typo in compile instructions
6be9e22 - Hannah Pierce-Hoffman, 3 days ago : compilation instructions for filename.sim
5751d7e - Kundan Guha, 4 days ago : Daisy, Daisy give me your heart to do...

60f87cf - Kundan Guha, 4 days ago : Added support for actually reusing outputs and inputs. Added new test case.
de12e0e - Kundan Guha, 4 days ago : Preliminary input loading working now
f89a45c - Kundan Guha, 5 days ago : Added support for constants to codegen
685f7b6 - Kundan Guha, 5 days ago : Started supporting external arguments in codegen
722cc18 - Kundan Guha, 5 days ago : Modified codegen to use arbitrary function names
3a860e4 - Yuanxia Lee, 5 days ago : Merge branch 'parser-shasha' of github.com:ultimatespirit/PLT-2017Fall into parser-shasha
c34905d - Yuanxia Lee, 5 days ago : Added hello world test to testparser.sh
9bb4f2e - Kundan Guha, 6 days ago : Added single line comments to last change (forgot about those)
7e87bb3 - Kundan Guha, 6 days ago : Added comment parsing to scanner, removed unused type program in ast
0c978f7 - Kundan Guha, 6 days ago : Removed commented out piece of code after testing previous commit
8e6b88d - Kundan Guha, 6 days ago : Testing removal of semicolon after gate_def
cb77493 - Yuanxia Lee, 6 days ago : Added gate to typ
6290640 - Kundan Guha, 6 days ago : Merge branch 'parser-shasha' of github.com:ultimatespirit/PLT-2017Fall into parser-shasha
54e121e - Yuanxia Lee, 6 days ago : Missing semi token
ffc839b - Yuanxia Lee, 6 days ago : Modified parser to handle gate declarations
daf7812 - Kundan Guha, 6 days ago : Potentially fixed previous gate declaration bug, needs testing
248a5ed - Kundan Guha, 6 days ago : Merge branch 'parser-shasha' of github.com:ultimatespirit/PLT-2017Fall into parser-shasha
5daa00c - Yuanxia Lee, 6 days ago : Merge branch 'parser-shasha' of https://github.com/ultimatespirit/PLT-2017Fall into parser-shasha
0ef26d6 - Yuanxia Lee, 6 days ago : Fixed merge conflicts
608a725 - Kundan Guha, 6 days ago : Added makefile changes from master into parser-shasha
f9b3d3e - Kundan Guha, 6 days ago : Forgot to add dependencies to the logisimple.native modification
d7e020c - Kundan Guha, 6 days ago : Merge branch 'master' into parser-shasha
598b4ec - Kundan Guha, 6 days ago : Modified Makefile to use menhir and not have logisimple.native be a phony rule
9e5d594 - Kundan Guha, 6 days ago : Fixed front end to compile
07910d6 - Kundan Guha, 7 days ago : Parser compiles fine now, however still small problems with stmts
3be7375 - Kundan Guha, 11 days ago : Fixed small bug introduced in last merge
64252fc - Kundan Guha, 11 days ago : Merge branch 'parser-shasha' of github.com:ultimatespirit/PLT-2017Fall into parser-shasha
3d95eb6 - Kundan Guha, 11 days ago : Parser lists fleshed out, added prims. Not working.
cbaaab9 - Yuanxia Lee, 11 days ago : Modified parser and created testparser.sh
6cc992a - Hannah Pierce-Hoffman, 11 days ago : git was telling me to add this so i did
2d86099 - Kundan Guha, 13 days ago : Finalising parser. Still has no provisions for array indexing.
e3a646d - Hannah Pierce-Hoffman, 13 days ago : and also removed hannah-sarah-tmp directory from parser-shasha branch, all of Hannah and Sarah's most up-to-date work can now be found on the master branch, src directory, will update again if this changes
85d6e02 - Hannah Pierce-Hoffman, 13 days ago : Merge branch 'master' of https://github.com/ultimatespirit/PLT-2017Fall
ea97268 - Hannah Pierce-Hoffman, 13 days ago : cleaning up master branch
cd4698a - Sarah Walker, 2 weeks ago : [tests] Cleaned up tests directory and added print lines to testall script
db2cde0 - Hannah Pierce-Hoffman, 2 weeks ago : builder for multiple primitives in netlist working but still needs some tweaks, solved principle issue of more than one prim in main function. note that stand-in parser and netlist have been slightly modified, nothing major. Run testall.sh to see current state of LLVM generated from each test.
21b2f57 - Yuanxia Lee, 2 weeks ago : Slightly modified ast and parser
fa58246 - Kundan Guha, 3 weeks ago : Small changes to ast and scanner. Made ast an mli file.
871ccf8 - Kundan Guha, 3 weeks ago : Parser not yet working, ast reworked

ba344cd - Sarah Walker, 3 weeks ago : [AND, OR, NOT, tests, cleanup] AND, OR, NOT working, but only as single expression per Logisimple program. Added test files and test script. Please run ./testall.sh to test all three types of gates and see llvm IR output.
729319b - Hannah Pierce-Hoffman, 4 weeks ago : just sorting out git issue but HELLO WORLD
3bddc54 - Sarah Walker, 4 weeks ago : HELLO WORLD in hannah_sarah_tmp
9c3edf5 - Hannah Pierce-Hoffman, 4 weeks ago : SOMETHING IS RUNNING AND COMPILING
f67e444 - Sarah Walker, 4 weeks ago : pushing all our changes, see fb messenger for details
4b4d41f - Hannah Pierce-Hoffman, 4 weeks ago : codegen on the correct branch, sorry, was confused
499a052 - Ultimatespirit, 4 weeks ago : Added sample Ring Oscillator C code
59f8c9e - Yuanxia Lee, 6 weeks ago : Modified to new syntax
05bb040 - Hannah Pierce-Hoffman, 6 weeks ago : the directory src/hannah_tmp contains my work on creating an extremely stripped down scanner, parser, and AST, next step is codegen, pushing if anyone wants to work on this more tonight
d791ec0 - Hannah Pierce-Hoffman, 6 weeks ago : Merge branch 'master' of https://github.com/ultimatespirit/PLT-2017Fall
f525b31 - Sarah Walker, 6 weeks ago : Created sample C program to model what tick is supposed to do
7acfb8e - Hannah Pierce-Hoffman, 6 weeks ago : hello world with updated syntax in hello.sim
5f1bb66 - Hannah Pierce-Hoffman, 6 weeks ago : helpfulcode.txt is a file with a summary of one of our meetings with prof Edwards
c545a07 - Ultimatespirit, 6 weeks ago : Fixed bug in scanner.mll
0b3c2d2 - Ultimatespirit, 6 weeks ago : Parser should be able to parse sample now
27ea4cd - Yuanxia Lee, 6 weeks ago : Added bool
4e43a1b - Yuanxia Lee, 6 weeks ago : Small changes to codegen
124bbcf - Hannah Pierce-Hoffman, 6 weeks ago : some declarations in codegen?
193475d - Yuanxia Lee, 6 weeks ago : Beginning codegen file
29a8a91 - Hannah Pierce-Hoffman, 6 weeks ago : empty semant file, come back later
d27364f - Ultimatespirit, 6 weeks ago : Fixed up AST to match up with parser
93045db - Hannah Pierce-Hoffman, 6 weeks ago : complete first draft of AST
45047a1 - Hannah Pierce-Hoffman, 6 weeks ago : binop
1bb4430 - Hannah Pierce-Hoffman, 6 weeks ago : added array type
a1ca7bc - Hannah Pierce-Hoffman, 6 weeks ago : another line of ast
ca45a6e - Hannah Pierce-Hoffman, 6 weeks ago : Merge branch 'master' of https://github.com/ultimatespirit/PLT-2017Fall
67360aa - Hannah Pierce-Hoffman, 6 weeks ago : skeleton of ast.ml
0302b11 - Sarah Walker, 6 weeks ago : Changed Makefile to more simple template from Calculator; changed comment in .mly file to /**/ instead of (**)
bf02716 - Hannah Pierce-Hoffman, 6 weeks ago : changed file extension on ast.ml
494865d - Yuanxia Lee, 6 weeks ago : Changed file extension
a3cff25 - Sarah Walker, 6 weeks ago : Added Makefile
79b2d29 - Ultimatespirit, 6 weeks ago : Potentially finished scanner, fixing parser now.
19a84e8 - Yuanxia Lee, 6 weeks ago : Added sampleprogram
f40eea0 - Ultimatespirit, 6 weeks ago : Initial copy of grammar1 file
d5a41f1 - Ultimatespirit, 7 weeks ago : Initial scanner, parse section copied from calc.
fe79ad7 - Ultimatespirit, 7 weeks ago : Changed Docs folder to docs (lowercase)
1f5a227 - Hannah Pierce-Hoffman, 7 weeks ago : src directory with four empty files
4ed0c55 - Ultimatespirit, 7 weeks ago : Removed duplicate files
ada5337 - Hannah Pierce-Hoffman, 7 weeks ago : made docs folder for everything that is not code
4291f61 - Ultimatespirit, 8 weeks ago : Initial LRM commit, files straight from papaeria doc
37be6d7 - Sarah Walker, 10 weeks ago : Grammar created on 10/14
2aedc40 - ultimatespirit, 4 months ago : Initial commit

5 Architectural Design



5.1 Scanner

The scanner, seen in scanner.mll, is responsible for tokenizing an input Logisimple .sim program. Tokenization includes the following actions:

- Disregards newlines, carriage returns, and whitespace
- Disregards comments (anything between %% %% or after a single %)
- Creates tokens from gate declarations "AND," "OR," "NOT"
- Creates tokens from boolean literals 0 and 1
- Creates tokens from:
  - { } to surround a gate definition
  - () to surround input parameters
    to create an array
  - ; to end a statement

5.2 Parser

The parser is responsible for building an AST. Each program is parsed as one all-encompassing gate definition, conventionally named "TICK". The components of a gate definition are a gate name, a list of inputs, a list of outputs, and a gate body. The gate body consists of a list of statements (some of which may involve defining and instantiating additional gates within the top-level TICK gate), ending with an "out" assignment which assigns the output of the entire program.

5.3 AST

The AST is the initial data structure that is made during the compilation process. Within the AST, the entire program is considered as a gatedef (as described above in the "Parser" section). Statements within the body of a gate definition may include defintions of user-defined gates, declarations of primitive or user-defined gates, variable declarations, and variable assignments. The AST is not guaranteed to be semantically correct until it has been traversed by the Semant.check function.

5.4 Semant

The semant file ensures that the AST we pass in from parser is semantically correct. Therefore it doesn't allow for redefining of custom gates, using gates that don't exist, checking for the right number of inputs to primitives, etc. It recursively calls checkgatedef to reach a base case where there is a gate def with no gate definitions and passes in a netlist type program to Flatten that builds and returns a netlist stmt list to be passed to codegen.

5.5 Flatten

Flatten is called by Semant to to take the intermediate representation of the AST and turn it into a properly flattened netlist so that all user-defined gates ultimately become just primitive types. Unfortunately, this functionality doesn't currently exist. However, the implementation would've been done through name mangling when Semant passes in a user-defined gate that hasn't been flattened already through recursive calls. We would've provided unique identifiers to each primitive that the custom gate consisted of by searching through a given stringmap for the custom gate definition and renaming those primitive, and then we would've connected the flattened gate's inputs to the larger gate (since every program written in our language is a giant gate). Eventually, once Semant finished recursing, a fully-formed netlist would've been created.

5.6 Netlist

The Netlist is the final data structure that is constructed before LLVM code generation begins. It consists of a list of Netlist.Expr.types, which represent AND, OR, or NOT primitive types. Each primitive type stores its own name and its inputs. Our intention in creating the Netlist data structure was to perform LLVM code generation on an argument that was as simple as possible. By the time a program has been converted into a Netlist, all user-defined gates have been deconstructed, reducing any program to a sequence of primitive operations.

5.7 Codegen

Codegen.ml performs the following actions on the Netlist data structure to create LLVM IR code:
- Takes in a list of primitive declarations and a list of variables (the Netlist)
- Allocates space for variables and constants in the primitive declarations
- Pattern matches the declarations to their LLVM operation
- Builds a list of IR instructions
- Builds global input and output arrays that can later be accessed by the linked C wrapper

6 Test Plan

Tests for primitives and standard library features.

6.1 Test-to-Pass

6.1.1 Syntax and Parsing Tests

To ensure all syntax features pass (e.g. comments, gate declarations). Listing of syntax and parsing tests which are intended to pass:

```
%% Test AND gate

Author: Sarah Walker %%


TICK(and_in1, and_in2)
{
    AND and_out(and_in1, and_in2);
    out = and_out;
}\\
```

Output of this test:

```
Sarahs-MacBook-Pro:demo sarahwalker$ ./compile_file.sh ../src/tests/pass/one_statement_and2.sim tester.c
../src/tests/pass/one_statement_and2 tester ../src/tests/pass/one_statement_and2.ll
Sarahs-MacBook-Pro:demo sarahwalker$ ./tester 0 0
0, 0 -> 0
Sarahs-MacBook-Pro:demo sarahwalker$ ./tester 0 1
0, 1 -> 0
Sarahs-MacBook-Pro:demo sarahwalker$ ./tester 1 0
1, 0 -> 0
Sarahs-MacBook-Pro:demo sarahwalker$ ./tester 1 1
1, 1 -> 1
```

```
%% Test OR gate

Author: Sarah Walker %%

TICK(or_in1, or_in2)
{
    OR or_out(or_in1, or_in2);
    out = or_out;
}
```

Output of this test:

```
Sarahs-MacBook-Pro:demo sarahwalker$ ./compile_file.sh ../src/tests/pass/one_statement_or2.sim tester.c
../src/tests/pass/one_statement_or2 tester ../src/tests/pass/one_statement_or2.ll
Sarahs-MacBook-Pro:demo sarahwalker$ ./tester 0 0
0, 0 -> 0
Sarahs-MacBook-Pro:demo sarahwalker$ ./tester 0 1
0, 1 -> 1
Sarahs-MacBook-Pro:demo sarahwalker$ ./tester 1 0
1, 0 -> 1
Sarahs-MacBook-Pro:demo sarahwalker$ ./tester 1 1
1, 1 -> 1
Sarahs-MacBook-Pro:demo sarahwalker$
```

```
%% Test include statement for standard library functions

Author: Sarah Walker %%

TICK(in1, in2)
{
    include(XOR.sim);
    XOR my_xor(in1, in2);
    out = my_xor;
}
```

```
%% Test one NOT gate

Author: Sarah Walker %%

TICK(not_in)
{
    NOT not_out(not_in);
    out = not_out;
}
```

```
%% Test three primitive statements in one file

Author: Sarah Walker %%


TICK(first_and_in1, first_and_in2, first_or_in1, first_or_in2, second_and_in1,
↪  second_and_in2)
{
    AND first_and(first_and_in1, first_and_in2);
    OR first_or(first_or_in1, first_or_in2);
    AND second_and(second_and_in1, second_and_in2);
    out = second_and;
}
```

```
%% Test two primitive statements in one file

Author: Sarah Walker %%


TICK(a, b, d, e){
    AND c(a,b);
    OR f(d,e);
    out = f;
}
```

```
%% Test two statements which share a variable

Author: Kundan Guha %%

TICK(a,b) {
    AND c(a,b);
    OR d(a,c);
    out = d;
}
```

```
%% Test a user-defined gate where names are reused in different scopes

Author: Kundan Guha %%


TICK(a, b, c) {
  AND d1(a,b);
  NOT d(d1);
  NAND(a,b){
    AND c(a,b);
    NOT d(c);
    out = d;
  }
  NAND e(a,b);
  out = e;
}
```

6.1.2 Logic Tests

To ensure logical operations are correct. To test this, we visually inspected the LLVM IR generated by codegen, and also verified by manually testing an entire truth table's worth of logic on each gate test.

6.2 Test-to-Fail

6.2.1 Syntax and Parsing Tests

To ensure syntax errors fail (e.g. redefining gates, using undefined variables). Listing of syntax and parsing tests which are intended to fail:

```
%% Tests instantiating a gate which does not exist

Author: Sarah Walker %%

TICK(in1, in2)
{
    DOESNOTEXIST output(in1,in2);
    out = output;
}
```

Output of this test:

```
Sarahs-MacBook-Pro:submit sarahwalker$ ./compile_file.sh tests/fail/gat
e_notexist.sim tester.c
tests/fail/gate_notexist tester tests/fail/gate_notexist.ll
Fatal error: exception Failure("Undefined gate: DOESNOTEXIST")
Undefined symbols for architecture x86_64:
  "_TICK", referenced from:
      _main in tester.o
ld: symbol(s) not found for architecture x86_64
clang: error: linker command failed with exit code 1 (use -v to see inv
ocation)
```

```
%% Test reusing a gate name (illegal)

Author: Sarah Walker %%

TICK(in1, in2)
{
    AND name(in1, in2);
    AND name(in1, in2);
    out = gate;
}
```

Output of this test:

```
Sarahs-MacBook-Pro:submit sarahwalker$ ./compile_file.sh tests/fail/red
ef_gate.sim tester.c
tests/fail/redef_gate tester tests/fail/redef_gate.ll
Fatal error: exception Failure("Cannot redeclare variable: AND name")
Undefined symbols for architecture x86_64:
  "_TICK", referenced from:
      _main in tester.o
ld: symbol(s) not found for architecture x86_64
clang: error: linker command failed with exit code 1 (use -v to see inv
ocation)
Sarahs-MacBook-Pro:submit sarahwalker$
```

```
%% Tests using a variable that has not been previously defined

Author: Sarah Walker %%

TICK(in1, in2)
{
    AND my_gate(in1, in3);
    out = my_gate;
}
```

7 Lessons Learned

7.1 Yuanxia Lee

I think much of what I learned was very well encapsulated by what Hannah and Sarah wrote. However, to reiterate and add my flair:

1) Start early: Even though in the beginning, the team was working on a reasonable timeline. We didn't plan enough for serious setbacks, and we could've avoided this by starting a lot earlier. Being on time wasn't enough, because the moment you aren't it's incredibly difficult to be on track. Think run-times, don't go for average case if you know your Big-O is much worse. This also applies to the tools you use in class, as learning functional programming and ocaml as well as reading the books at an earlier stage would've been incredibly helpful in feeling prepared to tackle the project.

2) Get help: Don't be afraid to look dumb. Appearing dumb is better than actual lack of knowledge. I was often too worried about showing how unprepared I was, and thinking I could just look everything up later. However, that was unhelpful for my understanding and for my team as that meant I couldn't contribute until I reached that understanding.

3) Teamwork: Everyone has different strengths, and I learned the importance of trust in each other. If

we can't trust or truly listen to our teammates, then it's difficult for all of us to feel like we have a greater goal (AKA this project) for us to believe in and work towards. It also would've been better if we understood more clearly what kind of expectations we had of each other and of what we wanted from this project/class from the onset.

7.2 Sarah Walker

My biggest lessons learned are:
1) Learn about the tools you will be using (i.e. functional programming). OCaml is the tool we used to build our entire compiler, so it was important to understand functional programming well. I had some frustrating moments when I knew I could have implemented something easily in my favorite imperative language, but I just did not have the OCaml knowledge to do it.

2) Don't be afraid to look at, and learn from past projects. In our initial brainstorming sessions, we tried a little bit too hard to make our language unique. It became so unique, at some points, that the uniqueness eclipsed the usefulness. Our syntax quickly became difficult to use (and difficult to parse). I failed to recognize that, though many projects were very "C-like," they had features that made them unique and interesting without making them convoluted.

3) Set expectations at the start of the project. I think the most difficult part for our team was understanding what we expected from each other. We formed the team without a lot of discussion about how much time we wanted to put in each week, how much each person was expected to contribute, or when we wanted deliverables. I often found myself unsatisfied with our progress without knowing why I was frustrated, but I think it was mostly because we were not always all on the same page about what our priorities were at any given time.

Overall, if I could do the project over again, I would focus on preparing more in the early stages. I would have met with the team after Homework 2 (the OCaml assignment) to make sure we were all on the same page, and give us an opportunity to learn from each other. I would have set a rough timeline of the entire project and outlined, at the very least, how much time each of us should be spending on our tasks each week. I would have spent more time in the beginning looking at old projects and understanding why the successful ones were great. I would not have been so afraid to do a project that was similar to most of the projects in the past. I like Logisimple, and I think this was a really interesting project, but sometimes I got lost in my desire to make something unique and ended up making the project more difficult than it needed to be.

7.3 Kundan Guha

From this project I learned the importance of communication with the rest of my team members. I often had ideas about how the project would look, but couldn't always articulate to the team exactly how that would look, meaning we weren't all on the same page. Moreover, it's difficult to communicate with your team or contribute if you have a lack of time due to psets. So start early and plan well kids!

7.4 Hannah Pierce-Hoffman

This project forced me to confront two of my deepest fears: asking for help, and working in a group. I learned a significant amount about both.

Asking for help: It's really important to ask for help EARLY and OFTEN. With a project of this level of complexity, no one will understand everything on the first try. I was initially reluctant to ask for help from my teammates, from the TAs, and from Professor Edwards because I was overwhelmed by the number of things I didn't know. Additionally, when I did ask for help, I often found myself pretending to fully understand the answer, and walking away from the interaction still confused. I learned that you can't be embarrassed to say you don't "get" something, and that you should keep asking questions until you do get it, even if you have to ask questions that you feel are ridiculously simple. This was something I had conceptually grasped prior to this semester, but this project drove it home on an emotional level.

Teamwork: People's work and communication styles vary enormously. Every member of our team had different strengths, but we often struggled to combine them in ways that would be beneficial to the project. Philosophically, I learned that being part of a team requires respecting each member's different perspective, rather than expecting everyone to conform to the same norm. There isn't a concrete way to accomplish this; it's more like a mindset that you need to maintain. Additionally, I learned that a team will work toward goals more effectively if every member has participated in setting those goals and knows why they are important. Professor Edwards described the ideal team as a "benevolent dictatorship," but I think the democratic aspect is much more valuable than this epithet implies. Each group member needs to feel like they have a say in the direction the project is going, rather than just being told what to do by the manager.

8 Appendix

8.1 Full Code Listing

8.1.1 scanner.mll

```
/* Authors: Kundan Guha, Yuanxia Lee */
{
open Parser
(* Get error messages with line numbers (format found in an online ocamllex tutorial)*)
let line_num = ref 1
exception Syntax_error of string
let syntax_error msg = raise (Syntax_error (msg ^ " on line " ^ (string_of_int
↪  !line_num)))
}

let digit = ['0'-'9']
let digits = digit+
let cap = ['A'-'Z']
let low = ['a'-'z']
let alpha = (cap | low)


rule token = parse
    [' ' '\t' '\r' ] { token lexbuf }
  | '\n' { incr line_num; token lexbuf }
  | '%' { singlcomment lexbuf }
  | "%%" { multicomment lexbuf }
  | "out" { OUT }
  | '0' { ZERO }
  | '1' { ONE }
  | "AND" { AND }
  | "OR" { OR }
  | "NOT" { NOT }
(*  | "DFF" { DFF } *)
  | "bool" { BOOL }
  | cap (cap | digit | '_')* as lit { GATE(lit) }
  | low (alpha | digit | '_')* as lit { ID(lit) }
(*  | cap (alpha | digit)* as lit { DECORATOR(lit) } *)
  | digits as lit { LITERAL(int_of_string lit) }
  | '{' { LCBRACE }
  | '}' { RCBRACE }
  | '[' { LSBRACE }
```

```
  | ']' { RSBRACE }
  | '(' { LPAREN }
  | ')' { RPAREN }
  | ';' { SEMI }
  | ',' { COMMA }
  | '=' { ASSIGN }
  | _ as lit { syntax_error ("Unrecognised token: " ^ (String.make 1 lit)) }
  | eof { EOF }

and singlcomment = parse
  '\n' { token lexbuf }
 | _   { singlcomment lexbuf }
and multicomment = parse
  "%%" { token lexbuf }
| _    { multicomment lexbuf }
```

8.1.2 parser.mly

```
/* Authors: Kundan Guha, Yuanxia Lee, Sarah Walker, Hannah Pierce-Hoffman */
%{
open Ast
%}

%token ONE ZERO COMMA OUT
%token RCBRACE LCBRACE RSBRACE LSBRACE RPAREN LPAREN
%token ASSIGN AND OR SEMI NOT BOOL
%token EOF
%token <string> ID GATE
%token <int> LITERAL

%start program
%type <Ast.gate_def> program

%%
program:
    EOF { {name = ""; inp = []; body = []; out=Arr [];} }
  | gate_def EOF { $1 }

gate_def:
    GATE LPAREN id_list RPAREN LCBRACE stmt_list OUT ASSIGN vexpr SEMI RCBRACE
      { {name = $1; inp = $3; body = $6; out = $9;} }

id_list:
    /* Nothing */ { [] }
  | nonempt_id_list { List.rev $1 }

nonempt_id_list:
    ID { [$1] }
  | nonempt_id_list COMMA ID { $3 :: $1 }

stmt_list:
    /* Nothing */ { [] }
  | stmtl { List.rev $1 } /* Entirely to allow left recursion */
```

```
stmtl:
    stmt SEMI { [$1] }
  | gate_def { [Gdef $1] }
  | typ ID LPAREN exprl RPAREN SEMI { [Vset ($2, $4); Vdec ($1 $2)] }
  | stmtl stmt SEMI { $2 :: $1 } /* Left recursive */
  | stmtl typ ID LPAREN exprl RPAREN SEMI { List.append [Vset ($3, $5); Vdec ($2 $3)] $1
  ↪  }
  | stmtl gate_def /*No SEMI*/ { (Gdef $2) :: $1 }
  /* Wasn't sure of a more elegant way to deal with that case */

stmt:
    typ ID { Vdec ($1 $2) }
  | ID LPAREN exprl RPAREN { Vset ($1, $3) }
  | ID ASSIGN vexpr { Assign (Symb $1, $3) }
  | ID LSBRACE num RSBRACE ASSIGN vexpr { Assign ((Deref ($1, $3)), $6) }

typ:
    GATE { fun x -> Gdec ($1, x) }
  | prim { fun x -> Pdec ($1, x) }
  | BOOL { fun x -> Bdec x }
  | BOOL LSBRACE num RSBRACE { fun x -> Barr (x, $3) }

prim:
    AND { And }
  | OR { Or }
  | NOT { Not }
/*  | DFF { Dff } */

num:
    ONE { 1 }
  | ZERO { 0 }
  | LITERAL { $1 }

vexpr:
    expr { $1 }
  | LSBRACE exprl RSBRACE { Arr $2 }

exprl:
    /* Nothing */ { [] }
  | expr { [$1] }
  | expr COMMA exprl { $1 :: $3 }

expr:
    ID { Name $1 }
  | ONE { Bit true }
  | ZERO { Bit false}
  | ID LSBRACE num RSBRACE { Index ($1, $3) }
```

8.1.3 ast.ml

```
(* Abstract Syntax Tree, printing functions *)

(* Authors: Kundan Guha, Yuanxia Lee *)
```

```
(* Variable names self-evaluate, arrays arbitrarily contain names or bits *)
type expr =
    Name of string
  | Bit of bool
  | Arr of expr list
  | Index of string * int

type prim = And | Or | Not (*| Dff*)

(* Type (prim or custom name), variable name *)
type vdecl =
    Pdec of prim * string
  | Gdec of string * string
  | Bdec of string
  | Barr of string * int

type iden =
    Symb of string
  | Deref of string * int

(* For semantic checking useful to know number of outputs expected *)
type gate_def = {
    name: string;
    inp: string list;
    body: stmt list;
    out: expr;
} and stmt =
    Assign of iden * expr
  | Gdef of gate_def
  | Vdec of vdecl
  | Vset of string * expr list

(* Variables split up into declaration and assignment even if not written that way *)


(*Begin attempt at pretty printing functions*)
let rec string_of_expr = function
    Name(s) -> s
  | Bit(true) -> "true"
  | Bit(false) -> "false"
  | Arr(exprs) -> "[" ^ String.concat "" (List.map string_of_expr exprs) ^ "]"
  | Index(s,i) -> s ^ " " ^ string_of_int i

let string_of_prim = function
    And -> "AND"
  | Not -> "NOT"
  | Or -> "OR"
  (*| Dff -> "DFF" *)

let string_of_vdecl = function
    Pdec(p,s) -> string_of_prim p ^ " " ^ s
  | Gdec(s1,s2) -> s1 ^ " " ^ s2
  | Bdec(s) -> s
  | Barr(s,i) -> s ^ " " ^ string_of_int i
```

```
let string_of_iden = function
    Symb s -> ("Symb " ^ s)
  | Deref (s, i) -> ("Deref " ^ s ^ "[" ^ string_of_int i ^ "]")

let rec string_of_inp = function
    [] -> ""
  | h::t -> h ^ (string_of_inp t)

let rec string_of_stmt = function
    Assign(s,e) -> (string_of_iden s) ^ "=" ^ string_of_expr e
  | Gdef(g) -> string_of_gate_def g
  | Vdec(v) -> string_of_vdecl v
  | Vset(s,e) -> s ^ " " ^ String.concat "" (List.map string_of_expr e)
and string_of_gate_def gdef =
    (gdef.name) ^ "\n" ^
    (string_of_inp gdef.inp)^ "\n" ^
    (String.concat "\n" (List.map string_of_stmt gdef.body))^ "\n"^
    (string_of_expr gdef.out) ^ "\n"
```

8.1.4 netlist.ml

```
(* Authors: Sarah Walker, Hannah Pierce-Hoffman, Kundan Guha *)

type op = And | Or

type arg =
  Name of string
| Bool of bool

type expr =
  Prim of op * arg * arg (* Oper, In1, In2 *)
| Neg of arg (* In1 (unary) *)

type stmt =
Expr of string * expr

type program = Program of string * string list * stmt list * string list
```

8.1.5 semant.ml

```
(* Authors: Kundan Guha, Yuanxia Lee *)

open Ast

module StringMap = Map.Make(String)
module N = Netlist

let check {name; inp; body; out} =
    let report_duplicate exceptf l =
        let rec helper = function
          n1 :: n2 :: _ when n1 = n2 -> raise (Failure (exceptf n1))
        | _ :: t -> helper t
        | [] -> () in
    helper (List.sort compare l) in
```

```
let add_inputs input map = List.fold_left
    (fun m n -> StringMap.add n (Bdec n) m) map input in

let merge_scopes local global =
    let mf k lv gv = match lv, gv with
      v, None -> v
    | None, v -> v
    | (Some v), (Some w) -> lv in
    StringMap.merge mf local global in

let build_output_list out =
    let name_to_str e =
        match e with
          Name s -> s
        | _ -> raise (Failure ("Non-identifier in output list")) in
    match out with
      Name s -> [s]
    | Arr el -> List.map name_to_str el
    | _ -> raise (Failure ("Non-identifier in output list")) in

(* Remember to setup curv with input variables of gate as Bdec *)
let rec check_gatedef body (extv, extg) (curv, curg) setl gpmap =
    let lookup_var n local global = try StringMap.find n local
    with Not_found -> try StringMap.find n global
    with Not_found -> raise (Failure ("Undeclared variable: " ^ n)) in

    let lookup_gate n = try StringMap.find n curg
    with Not_found -> try StringMap.find n extg
    with Not_found -> raise (Failure ("Undefined gate: " ^ n)) in

    let check_deref n i local global =
      let var = lookup_var n local global in
      match var with
        Barr (nam, len) -> if i < len && i > -1 then var else raise (Failure
        ("Illegal dereference: " ^ nam ^ ", " ^ n))
      | (Pdec (_, _) | Gdec (_, _) | Bdec _) -> raise (Failure (
          "Illegal dereference: " ^ n ^ ", " ^ string_of_int i)) in

    let rec check_expr e =
        match e with
          Name s -> lookup_var s curv extv; e
        | Index (s, i) -> ignore (check_deref s i curv extv); e
        | Arr el -> List.iter (fun x -> ignore (check_expr x)) el; e
        | Bit _ -> e in

    let rec get_numouts vd =
        match vd with
          Pdec (_, _) -> 1
        | Gdec (n, _) -> (match (lookup_gate n).out with
            Bit _ -> 1
          | Index (_, _) -> 1
          | Arr el -> List.length el
          | Name s -> get_numouts (lookup_var s curv extv))
```

```
    | Bdec _ -> 1
    | Barr (_, i) -> i in

(* Checks types for assignment only *)
let check_type lhs rhs errf =
    match lhs with
      Bdec _ -> (match rhs with
        Bit _ -> ()
      | Index (_, _) -> ()
      | Name n -> if (get_numouts (lookup_var n curv extv)) == 1 then ()
        else raise errf
      | Arr el -> if (List.length el) == 1 then () else raise errf
      )
    | Barr (_, n) -> (match rhs with
        Arr el -> if (List.length el) == n then () else raise errf
      | Name na -> if (get_numouts (lookup_var na curv extv)) == n then
          () else raise errf
      | (Bit _ | Index (_, _)) -> raise errf)
    | (Pdec (_, _) | Gdec (_, _)) -> raise errf in

let check_stmt s setlist =
    let check_no_name n map err =
        if StringMap.mem n map then raise err else n in

    match s with
      Assign (id, expr) -> let rhs = check_expr expr in
        let var = (match id with
          Symb sy -> lookup_var sy curv StringMap.empty
        | Deref (sy, i) -> check_deref sy i curv StringMap.empty) in
        ignore (check_type var rhs (Failure ("Incompatible assignment: " ^
        string_of_stmt s))); (s :: setlist, curv, curg, gpmap)
    | Vdec vd -> let name = check_no_name (match vd with
        Pdec (_, s) -> s
      | Gdec (g, s) -> ignore (lookup_gate g); s
      | Bdec s -> s
      | Barr (s, _) -> s
      ) curv (Failure ("Cannot redeclare variable: " ^ string_of_stmt s)) in
        (setlist, StringMap.add name vd curv, curg, gpmap)
    | Gdef gd ->
        (try ignore (check_no_name gd.name curg (Failure (""))))
        with failure -> ignore (check_no_name gd.name extg (Failure (
            "Cannot redeclare gate: " ^ gd.name))));
        let gate = check_gatedef gd.body
        (merge_scopes curv extv, merge_scopes curg extg) (add_inputs
        gd.inp StringMap.empty, StringMap.empty) [] gpmap in
        (setlist, curv, curg, StringMap.add gd.name (
        N.Program (gd.name, gd.inp, gate, build_output_list gd.out)) gpmap)
    | Vset (id, el) ->
        let var = lookup_var id curv StringMap.empty in
        let err = Failure ("Incorrect number of arguments for gate: "
        ^ id) in
        (match var with
          Pdec (p, _) ->  (match p with
            (And | Or ) -> if List.length el != 2 then raise err
```

```
            | Not -> if List.length el != 1 then raise err )
          | (Bdec _ | Barr (_,_)) -> raise err
          | Gdec (g, _) -> if (List.length (lookup_gate g).inp) != List.length
          el then raise err); (s :: setlist, curv, curg, gpmap)
      in


      match body with
        [] -> Flatten.flat setl curv curg (*Actual interface TBD *)
      | h :: t -> let (sl, cv, cg, gm) = check_stmt h setl in
          check_gatedef t (extv, extg) (cv, cg) sl gm

      in
      match name with
        "" -> raise (Failure ("Empty file passed in"))
      | _ ->
          let gate = check_gatedef body
          (StringMap.empty, StringMap.empty) (add_inputs
          inp StringMap.empty, StringMap.empty) [] StringMap.empty in
          N.Program (name, inp, gate, build_output_list out)
```

8.1.6 flatten.ml

```
(* Beginnings of Flatten *)

(* Authors: Hannah Pierce-Hoffman, Kundan Guha, Yuanxia Lee*)

(* Open Ast and Netlist modules so we can use their data structures *)
module A = Ast
module N = Netlist

module StringMap = Map.Make(String)

(* Initialize empty list to build the netlist *)
let nl = []

(* Helper function to convert AST prims to Netlist prims.
Need to separately deal with Not/Neg conversion case later *)
let prim_converter aop =
    match aop with
        A.And -> N.And
      | A.Or -> N.Or

let expr_converter e =
    match e with
        A.Name s -> N.Name s

(* Arguments:
vset_assign_list is a list of A.Vset and A.Assign types
vdecmap is a StringMap of names to A.Vdec types
gatemap is a StringMap of names to netlist-style gate definitions
(you only need to look something up in gatemap if it is a user-defined
gate and not a primitive)
*)
```

```
let flat vset_assign_list vdecmap gatemap =

    (* The vs_lookup function takes a single A.Vset item and
    looks up its name in vdecmap.  If the name is found to
    represent a prim, a list containing a single netlist-ready N.Expr.Prim
    object is returned.  If the name is found to represent a user-defined
    gate, we look up the gate in gatemap, then return
    the relevant list of netlist definitions that represent this
    user-defined gate.  In either case, vs_lookup returns a list of
    netlist-ready statements that can then be concatenated with the
    growing netlist. *)
    let vs_lookup vs =
        match vs with
            A.Vset(vname, inp_list) ->
                let found_vdecl = StringMap.find vname vdecmap in
                match found_vdecl with
                    A.Pdec (prtype, p_vname) ->
                        match inp_list with
                        [in1; in2] ->
                            let nlop = prim_converter prtype in
                            (* Possible concern: types of in1 and in2 *)
                            let nl_ready_stmt = N.Expr(vname, N.Prim(nlop, expr_converter
                            ↪  in1, expr_converter in2))
                        in [nl_ready_stmt]
                        | [in1] ->
                            let nl_ready_stmt = N.Expr(vname, N.Neg(expr_converter in1))
                        in [nl_ready_stmt]

                  (* For Gdec, the gatebody is assumed to be a list of netlist-ready
                  ↪  statements,
                  so we just return this and don't worry further about the inputs.
                  | A.Gdec (custom_gtype, g_vname) ->
                        let found_gatebody = StringMap.find vname gatemap in
                        found_gatebody *)

    (* Call vs_lookup on every item in vset_assign_list using List.map and save the end
    ↪  result
    in nl_ready_list*)
    in
    let list_of_lists = List.map vs_lookup vset_assign_list in
    List.concat list_of_lists

    (* Desired result: nl_ready_list is a list of N.Expr, ready for codegen *)
```

8.1.7 codegen.ml

```
(*===----------------------------------------------------------------------===
 * Code Generation
 *===----------------------------------------------------------------------===*)

(* Authors: Kundan Guha, Hannah Pierce-Hoffman, Sarah Walker*)

module L = Llvm
module A = Netlist
```

```
module StringMap = Map.Make(String)

let translate (A.Program (nam, inputs, prims, outputs)) =

    let context = L.global_context () in
    let the_module = L.create_module context "Logisimple"
    and void = L.void_type context
    and i8_t = L.i8_type context
    and i1_t = L.i1_type context in
    let i8p_t = L.pointer_type i8_t
    and zero = L.const_int i8_t 0
    and one = L.const_int i8_t 1 in
    let extargs = [ "Inp"; "Out" ] in

    let ftype = L.function_type void (Array.make (List.length extargs) i8p_t) in
    let mainf = L.define_function nam ftype the_module in
    let builder = L.builder_at_end context (L.entry_block mainf) in
    let formals =
        let make_formals l n p =
            L.set_value_name n p;
            let local = L.build_alloca i8p_t n builder in
                ignore (L.build_store p local builder);
                local :: l in
        List.rev (List.fold_left2 make_formals [] extargs (Array.to_list (L.params
        ↪  mainf)))
    in
    let ext_vars =
        let build_input nam count map =
            let arg = L.build_alloca i8_t nam builder
            and arrp = L.build_load (List.hd formals) "" builder in
            let arr = L.build_in_bounds_gep arrp [| L.const_int i8_t count |] "input"
            ↪  builder in
            let arr1 = L.build_load arr "" builder in
            ignore (L.build_store arr1 arg builder);
            StringMap.add nam arg map in
        let rec build_inputs count l map =
            match l with
              [] -> map
            | h :: t -> build_inputs (count + 1) t (build_input h count map) in
        build_inputs 0 inputs StringMap.empty in
    (*let rec print_keys l = match l with
      [] -> print_endline "contained"
    | (x, y) :: t -> print_bytes (x ^ " "); print_keys t in
    ignore (print_keys (StringMap.bindings vars));*)
    let lookup n vars = try StringMap.find n vars
                  with Not_found -> print_endline ("Key: " ^ n ^ " not found.");
                  raise Not_found in
    let codegen_arg a vars builder =
        match a with
          A.Name n -> L.build_load (lookup n vars) n builder
        | A.Bool b -> if b then one else zero
    in
    let expr builder vars (A.Expr (out, gate)) =
```

```
        let store_output = L.build_alloca i8_t out builder in
        (match gate with
          A.Prim (op, in1, in2) ->
            let pt1 = codegen_arg in1 vars builder and
                pt2 = codegen_arg in2 vars builder in
                let and_result =
                    (match op with
                        A.And -> L.build_and
                        | A.Or -> L.build_or
                    ) pt1 pt2 "tmp" builder in
                    ignore (L.build_store and_result store_output builder);
        | A.Neg in1 ->
            let pt5 = codegen_arg in1 vars builder in

            let not_result = L.build_not pt5 "tmp" builder in let not_result =
                L.build_and not_result one "tmp" builder in
            ignore (L.build_store not_result store_output builder);
        );
        StringMap.add out store_output vars
    in
    let nlist vars (e: A.stmt) = expr builder vars e
    in
    let all_vars = List.fold_left nlist ext_vars prims in
    let output =
        let build_output nam count map =
            let arg = L.build_load (lookup nam map) nam builder in
            let arrp = L.build_load (List.hd (List.tl formals)) "" builder in
            let arr = L.build_in_bounds_gep arrp [| L.const_int i8_t count |] "output"
            ↪  builder in
            ignore (L.build_store arg arr builder); map in
        let rec build_outputs count l map =
            match l with
              [] -> ()
            | h :: t -> build_outputs (count + 1) t (build_output h count map) in
        build_outputs 0 outputs all_vars in
    ignore (L.build_ret_void builder);
    the_module
```

8.1.8 logisimple.ml

```
(* Authors: Kundan Guha, Hannah Pierce-Hoffman *)

module StringMap = Map.Make(String)

type action = Netlist | LLVM_IR | Compile

let _ =
  let action = ref Compile in
  let set_action a () = action := a in
  let speclist = [
    ("-a", Arg.Unit (set_action Netlist), "Print the SAST");
    ("-l", Arg.Unit (set_action LLVM_IR), "Print the generated LLVM IR");
    ("-c", Arg.Unit (set_action Compile),
      "Check and print the generated LLVM IR (default)");
```

```
  ] in
  let usage_msg = "usage: ./logisimple.native [-a|-l|-c] [file.sim]" in
  let channel = ref stdin in
  Arg.parse speclist (fun filename -> channel := Unix.open_process_in ("m4 " ^ filename))
  ↪  usage_msg;
  let lexbuf = Lexing.from_channel !channel in
  (* Scan and then parse *)
  let ast = Parser.program Scanner.token lexbuf in
  let the_nl = Semant.check ast in
  let m = Codegen.translate the_nl in
  Llvm_analysis.assert_valid_module m;
  print_string (Llvm.string_of_llmodule m)
```

8.1.9 tester.c

This tester file is the "wrapper" that must be compiled and linked with each Logisimple program to allow for circuit simulation.

```
#include <stdio.h>

/* Authors: Kundan Guha, Hannah Pierce-Hoffman, Sarah Walker */

char TICK(char*,char*);
int main(int argc, char ** argv) {
    if (argc < 2 ) return -1;

    char inp[10];
    int index = 1;

    while (index < argc)
    {
        inp[index - 1] = argv[index][0] - '0';
        index = index + 1;
    }

    char out[1];
    TICK(inp,out);

    for (int j = 0; j < argc - 2; j ++)
    {
        printf("%d, ", inp[j]);
    }
    printf("%d ", inp[argc - 2]);
    printf("-> ");
    printf("%d\n", out[0]);

    return 0;

}
```