Scirch

Circuit Simulation Language

By:
Jeff Sinckler
Brian Hunter

Problem Statement

Circuits are tedious to analyze.
How can we make this a little bit easier?

The inspiration for Scirch

Scirch (The Ideal Version)

- Scirch is a Circuit Simulation Language.
- Scirch allows users to build individual circuits.
  - These circuits can be used as components for the main circuit.
- Inputs are able to be named and passed into these circuits.
- Return values from circuit components can also be stored and passed into other circuits.
- Basic gates (and, or, not, nand, nor) are built into the language and provided with special characters.
- Multiplexers and Encoders are also built into the language and provided to users.
- Testing functions are provided to users
  - Pass in a circuit and see results for all possible inputs to the circuit.

Scirch (What actually exists)

- Basic code compiler.
- Allows users to create functions (which are supposed to represent circuits)
  - Functions return values corresponding with the last calculation performed.
- Store outputs of gates in variables.
  - Call/use those variables in other logic calculations.
- Print outputs of values to the console.
- Generate and print bytecode listing.
- Generate and run bytecode.
- No types. All variables hold integers.

Scirch Syntax

- Scirch is not interactive; source code must be placed in a file.
- Main function declared inside curly braces
  - { *main function text here; }*
- Other functions declared before main function
  - Functions declared with a *name* followed by *curly braces*.
  - sampleFunction{ *function body here;* }
- Functions called using parenthesis
  - { sampleFunction(); } </ Calling sampleFunction() in the main function />

Scirch Syntax

- Variables declared in functions, in the main, or globally.
  - sampleFunction{ declare sampleVar; } </ Variable declared in function />
  - { sampleVar = 1; } </ Variable declared in main function />
  - If a variable is being initialized without a value, use the declare keyword.
  - If a variable is being set, do not include the declare keyword.
- Semicolons separate statements
  - sampleFunction{ *stmt1; stmt2; stmt3; }*
- Comments are placed between </ and />
  - </ This is a comment line! />

Scirch Syntax

- Variables declared globally are defined outside of any function.
  - sampleFunction{0^0; sampleVar = 0;}
    declare sampleVar;
    {sampleVar = 1; sampleFunction(); print(sampleVar);}
  - All functions, including sampleFunction() can access and modify sampleVar.
  - Allows multiple values to be returned from a function.

Running Scirch

- The command line scirch executable takes two arguments.
  - Arg1: Selects what exactly you want Scirch to do with your code.
  - Arg2: Specifies the location of the source file that you want Scirch to work with.
- Command line syntax:
  - ./scirch [-i | -c | -e] <file path>
  - The arguments cannot be ignored. Must be run with both an option and a filename.

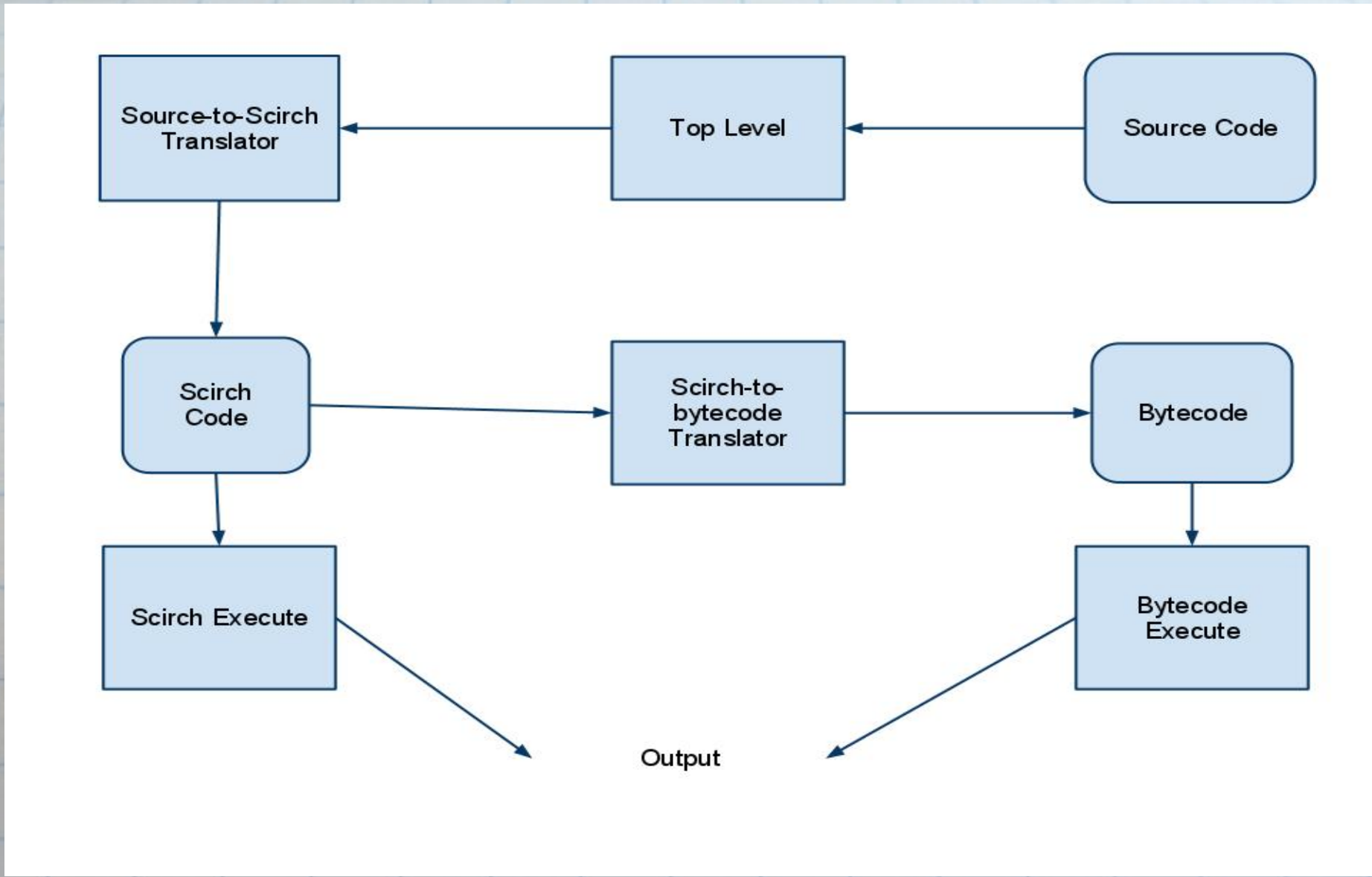
Scirch Arguments and Features

- **-i -- Interpret** - Translate the source into native Scirch code using the abstract syntax tree and execute.
- **-c -- Compile** - Translate the source into native Scirch code and translate that into faux bytecode. Print the bytecode.
- **-e -- Execute** - Translate the source into bytecode and execute the bytecode.

File Breakdown

File Name	Number of Lines	Purpose
gates_scanner.ml	25	Provides the tokens needed in order to parse source files.
gates_parser.ml	53	Context free grammar that states how to reduce lines.
gates_ast.ml	32	Abstract syntax tree that provides types used in the background by scirch.
gates_main.ml	33	Holds the functions that perform appropriate actions on lists of translated source code.
firstgates2.ml	20	Library file that holds the functions for the five basic gates in Scirch.
compile.ml	29	Holds the code that translates a list of commands into bytecode.
execute.ml	32	Holds the functions that perform the appropriate actions on the bytecode generated from the source list
bytecode.ml	13	Abstract syntax tree that provides types used in the background by scirch after translating to bytecode.
scirch.ml	32	Top level of scirch.

How it works in Ocaml



Summary

- In making Scirch, we made many decisions with realistic circuit development in mind.
- Scirch doesn't have everything imagined, but still worked out to be an acceptible compiler that translates text into executable code of different forms.
  - Learned Ocaml (functional programming languages)
  - Experienced Lambda Calculus
  - Learned how bytecode works and how a compiler can (possibly) translate it.
    - Gives a different perspective on how to program, what programs can do, and how to make them do it.