Frost HDL Compiler Scopes Implementation

Andrew Clark

February 27, 2019

Table of Contents

Table of Contents **1**

Scope Types and Containable Ones, Too **2**

Overall Handling of Scopes **3**

Types of Scopes **3**

4.1 `modules` 3

Scope Types and Containable Ones, Too

- `module`
  - `struct` / `class`
  - `enum`
  - `function` / `task`
  - Behavioral code blocks
    - `initial`
    - `always`
    - `always_comb`
    - `always_seq`
    - `generate`
  - `module` instantiations
- `package`
  - (Other) `package`
  - `struct` / `class`
  - `enum`
  - `function` / `task`
- `struct` / `class`
  - (Other) `struct` / `class`
  - `enum`
  - `function` / `task`
- `enum`
- `function` / `task`
- Behavioral code blocks
  - `initial`
  - `always`
  - `always_comb`
  - `always_seq`
  - `generate`

Overall Handling of Scopes

Types of Scopes

4.1 `modules`

This is the primary type of scope, without which no Verilog code will be generated. The other types of scopes are intended to be used as helpers for this type of scope. These are analagous to Verilog `module`s.

`parameter`ized `module`s are extremely useful, and will be supported directly. However, they may generate Verilog code for non-`parameter`ized `module`s so that Frost HDL can use its own semantics for them.

A `module` can only be placed at global scope, at least in the initial version of the language. Because `parameter`ized `module`s are probably going to have their names mangled, it will probably be easy to allow `module` definitions in scopes other than global scope. If `module` definitions are ever allowed in non-global scopes, such scopes will probably have to be either a `package` or another `module`.