# Frost HDL

Andrew Clark (FL4SHK)

2018-11-03

## Abstract

Eggs!

# Table of Contents

# Introduction

- This is a "high level" hardware description language. The main goal is to take features from synthesizeable SystemVerilog (which aren't in Verilog) and expand upon them, and also to have direct support for yosys's formal verification abilities.
- The language syntax is also set up differently from both Verilog and SystemVerilog to eliminate some of my pet peeves.
- This language compiles to Verilog-2001, but support for compiling to SystemVerilog may be added later. On that note, this language is ONLY compiled, and does not have its own simulator. This allows me to piggy back off of Verilog's semantics.

# Language Details/Features

- Syntax changes
  - The biggest syntax change is that `{` and `}` are used to indicate code blocks, **not** `begin` and `end`. I will also have `{` and `}` be used to indicate `module` contents, so `endmodule` is gone.
  - As such, concatenation needs new syntax. I have simply chosen `$concat()` for the new syntax.
  - Replication also has a new syntax: `$replicate()`.
- What doesn't exist
  - Verilog's `fork` and `join` constructs are not supported at this time, but may be added later, especially if I can figure out what they're actually used for. They do not appear to be synthesizeable constructs anyway.

- `package`s
  - Nested `package`s are supported. This may be something that SystemVerilog proper supports, but Icarus Verilog's implementation of SystemVerilog's `package`s definitely does not seem to do so.
- `struct`s
  - First and foremost, unlike in SystemVerilog, `struct`s can be `parameter`ized.
  - A `struct` can have member `task`s and member `function`s. These operate basically how you would expect.
  - `public` and `private` are supported as well.
  - Constructors and destructors are not supported. Destructors in particular do not make sense to support, but constructors **may** be added for `struct`s that end up implemented as `reg` vectors in the generated code.
  - For code generation, `struct`s are simply compiled to either `wire` or `reg` vectors, with a `struct`'s data members simply being different slices of the generated `wire` or `reg` vector.
  - Note that `struct`s in this language are very similar to SystemVerilog's `packed struct`s, and SystemVerilog's non-`packed struct`s simply do not exist.
- `union`s
  - `union`s may or may not be added. I honestly **might** add them, but it's not as high a priority as other things.
- `interface`s
  - These are intended to mimic SystemVerilog's `interface`s, though I may make some changes.

  - Having not had much access to SystemVerilog's `interface`s myself, I may have to review how they work first before I can implement them.

- `macro`s
  - I really want to support a useful macro system, perhaps in the form of a preprocessor. This is perhaps my replacement for Verilog's `generate`, but I might need to support `generate` as well.

- Formal Verification
  - yosys's formal verification constructs are directly supported and will simply be inserted directly into the generated code.
  - The constructs `assert()`, `assume()`, `assert property`, `assume property`, etc. are all supported directly.