Icarus Verilog Status and Goals

Stephen Williams

Creator and principal developer for Icarus Verilog.
<steve@icarus.com>
<http://iverilog.icarus.com>

A Word on Applicability

- Open Hardware is caged in without tool interoperability.
- Standards help.
- Open implementations of standards free one from vendor lock-in.
- Open source tools provide (paradoxically) better stability.

Outline

- License Issues
- Current Status
- Development plans
- Support
- Conclusions

Icarus Verilog License

- Open Source (GPL)
  - The compiler/translator itself is licensed using GPL licenses
- Proprietary plug-ins supported
  - Plug-ins (PLI, ivl_target) are licensed separately
  - Proprietary plug-ins are allowed by explicit statement in the license terms to Icarus Verilog

What is Icarus Verilog

- HDL Simulator
  - Simulates standard HDL descriptions
  - Includes standard C/C++ interfaces
- HDL Translator
  - Includes an ivl_target API for access to elaborated design
  - ivl_target API can be used by plug-ins
- Limited Synthesis
  - Some targets make use of the internal synthesis capabilities.

Relevant Standards

- IEEE 1364 (Verilog)
- IEEE 1800 (SystemVerilog)
- IEEE 1076 (VHDL)
- Accellera VAMS (Verilog A/MS)
- Others for code generators

Verilog Status

- IEEE 1364-2005
- Most of the language is supported
- Widely used
- Active development

System Verilog Status

- IEEE 1800
- Significant level of support
- High demand, but...
- Not yet widely used (too new)
- Active development

VHDL Status

- IEEE 1076
- Limited support
- Increasing demand
- Active development

Verilog A/MS Status

- Accellera Standard
- Very limited support
- Limited demand

Verilog Plans

- Implement missing language features
- Track standards variants
- Improve simulation engine (vvp)

System Verilog Plans

- Implement missing language features
- Track standards
- SV is the new Verilog

VHDL Plans

- Implement synthesizable subset
- Define Verilog/VHDL interaction
- VHDL-specific run-time support

Other Plans

- Improve limited synthesis capabilities
- Plug-ins to perform interesting design verifications
- Performance improvements

Code Generators

- VVP (general simulation)
- VHDL (to translate to VHDL)
- Verilog (to translate to simplified Verilog)
- Proprietary plug-ins allowed

More Information

- Icarus Verilog web sites
  - <http://iverilog.icarus.com>
- Development mailing list
  - iverilog-devel@sourceforge.net
- GIT Repository (github)
  - https://github.com/steveicarus/iverilog

Related tools

- GTKWave - Standalone waveform Viewer
  - Supports high performance dumper formats
  - <http://gtkwave.sourceforge.net>
- SIMBUS – System Level simulation aid
  - Divides simulations into networked processes
  - <http:iverilog.wikia.com/wiki/SIMBUS>
- Others?

“Competitor” Tools

- GPL Cver
  - Seems to be no longer actively supported
  - <http://sourceforge.net/projects/gplcver/>
- Verilator
  - Limited language support, but
  - Fast at what it does (references?)
  - <http://www.veripool.org/wiki/verilator/>
- ghdl
  - Only other open source VHDL
  - Not widely ported
  - <http://ghdl.free.fr/>

How to Participate

- Submit well written bug reports
- Contribute patches
  - Cadre of developers act as gate keeper
- Contribute code generators/plug-ins
  - Need not be bundled with IV
  - Can be proprietary
- Regression tests
  - <https://github.com/steveicarus/ivtest>

How to Contribute

- Direct Donations
  - Allows us to purchase infrastructure to support development.
  - Gives us free reign to perform work that doesn't have obvious or immediate visible impact.
- Contract Development
  - Allows contributors to sponsor specific tasks.
  - Customer can negotiate schedule, scope, and completion criteria.

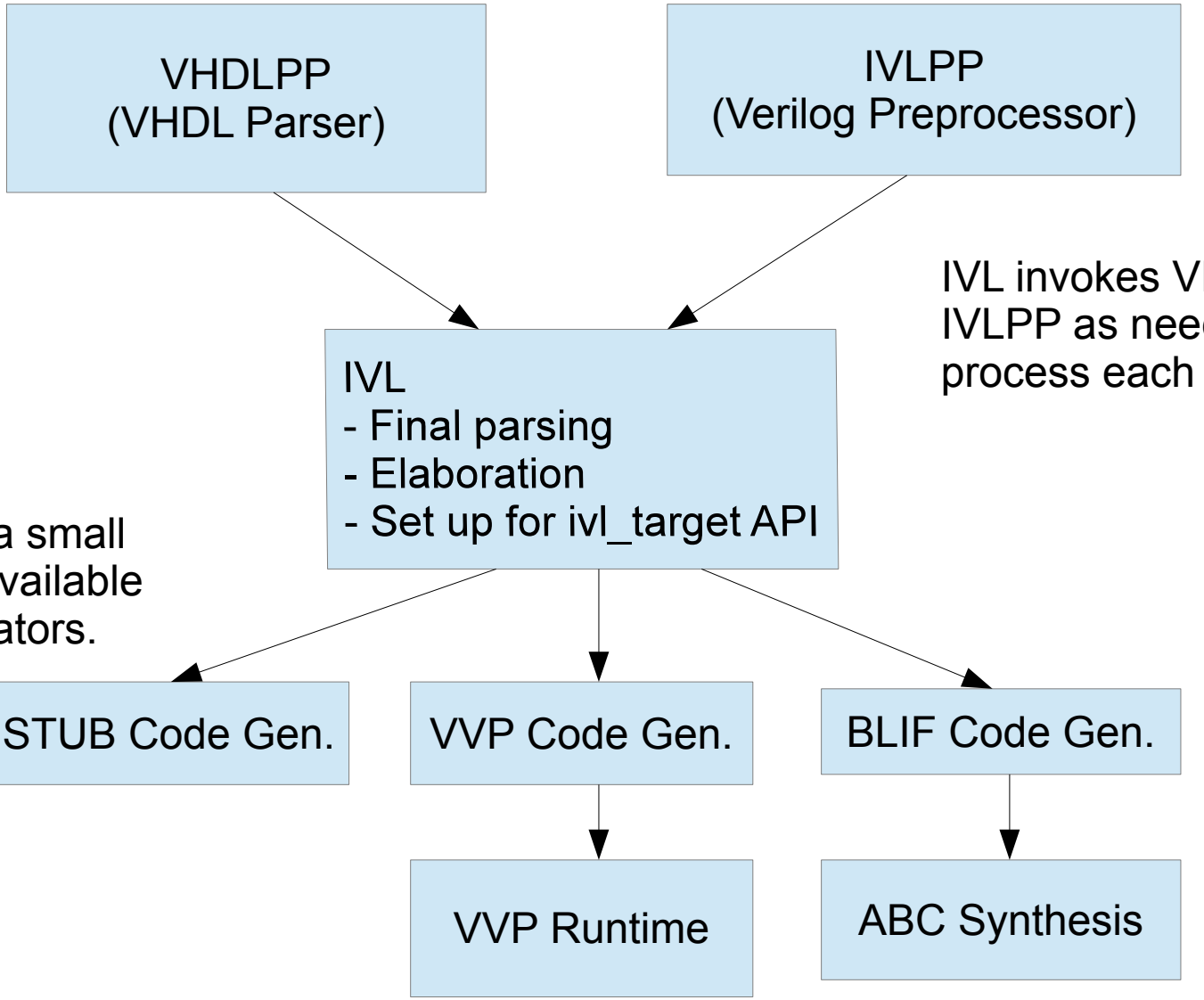
Conclusion - Icarus Verilog is Awesome

- It is widely used, professionally and by hobbyists
- Community support is available
- Professional support options (by individuals) are available
- Great value
- (Not perfect.)

Icarus Verilog Internals Overview

A very brief overview of how the Icarus Verilog suite of tools works. This should give you an idea how the major components of the compiler interact, and why.

General Structure (simplified)



IVL invokes VHDLPP or IVLPP as needed to process each input file.

This is just a small sample of available code generators.

Iverilog driver

- The “iverilog” command is actually a driver that invokes compiler parts.
- The components are installed in the library directory <*libdir*>/ivl/*
- The driver communicates with the sub-programs via command lines and config files.
- The driver takes the command line and config files, and invokes the necessary parts of the compiler.

Input File Streams

- IVL program gets list of input files from the driver:
  - VHDLPP is invoked to read VHDL files
  - IVLPP is invoked to read Verilog files.
- IVL takes in and parses blended input stream, which is extended Verilog.

Elaboration

- IVL elaborates the blended input stream
  - Names are resolved
  - Parameters are defined/completed
  - “generate” scopes are generated
- Scopes are finalized, Modules are instantiated
- Elaborated design prepared for code generation

Elabortion (cont.)

- Elaboration may invoke ivlpp or vhdlpp to prong in library modules
- Packages are also managed by elaboration.

Code Generation

- User-selected code generator is invoked
- Code generators invoked by IVL as plug-ins.
- The standard simulation (vvp) is targeted by a code generator plug-in.

Code Generation (Cont.)

- <ivl_target.h> API defined for code generators.
- <*libdir*>/ivl/*.conf defines processing that the code generator expects of ivl (i.e., synthesys, etc).
- <*libdir*>/ivl/*.tgt files are the code generator plug-in files.

Simulation

- VVP program simulates compiled design
- VVP code generator (-tvvp) generates input for VVP program
- Input is ASCII text, but meant for machine reading (Think: assembler)
- Verilog VPI is supported by vvp engine

Simulation – VPI/PLI

- VPI Modules loaded from <*libdir*>/ivl or user specified location
- Files *.sft define run-time types for system functions
- Files *.vpi are the compiled VPI modules

Simulation – Waveform Dumps

- The bundled $dumpvars() function supports Verilog VCD for portability.
- We also support high-efficiency dumper formats that are more compact
- GTKWave is a separate tool that displays VCD dumps and also the high-efficiency dumper formats.

Other Code Generators

- <ivl_target.h> API defines API for custom code generators.
- Users may install custom generators by building and installing *.conf and *.tgt files.
- Example code generators:
  - VVP (simulation)
  - BLIF (ASIC logic synthesis using ABC)
  - Verilog-95 (translates to simplified Verilog)
  - VHDL (translates to VHDL)

Conclusion – Icarus Verilog is Expandable

- Compilation/Elaboration separated from code generation
- Code generation target can be selected on command line
- Code generators can be packaged separately.
- API for developing code generators is relatively stable