

# Verifying Verilog modules with Ruby-VPI

Suraj N. Kurapati

UC Santa Cruz

October 17, 2006

# Agenda

- 1 Background
  - Concepts
  - Verilog
  - Ruby
  
- 2 Introduction
  - Usage
  - Mechanics
  - Interface

# What is verification?

- You have something
- You want to check if it works
  - Stimulus and response

## Example

You install a new doorbell in your house.  
You press it and expect to hear it ring.

# Verification terms

**Design** The thing being checked: does it work or not?

**Expectation** Desired response to some stimulus.

**Specification** Your expectations about the design.

**Test** Checks if the design satisfies the specification.

**Bench** Place where the design is tested.

# What is VPI?

## Definition

VPI — Verilog Procedural Interface

- Bridge between Verilog and C
  - Verilog can invoke C functions
  - C can inspect and modify<sup>1</sup> Verilog
- Verilog can do anything C can

---

<sup>1</sup>Cannot create new Verilog objects.

# Verilog and C interaction

- Verilog code invokes C functions when necessary
  - C functions are passed signals to check
  
- Simulator invokes C functions upon certain events
  - C functions inspect signals of interest
  - Asynchronous, event-driven programming

# How does VPI work?

- 1 You compile C code into an object file
- 2 You run Verilog simulator with object file
- 3 Simulator invokes functions in `vlog_startup_routines` array
  - 1 Create new Verilog tasks
  - 2 Associate them with C functions
- 4 Simulator simulates Verilog code
  - 1 Verilog code invokes tasks created by object file
  - 2 Simulator invokes associated C functions instead

# What about PLI?

## Definition

PLI — Procedural Language Interface

- Ancestor of VPI with common goal
- TF and ACC deprecated in IEEE Std. 1364-2005

Interface	Generation	No. functions
TF	1	65
ACC	2	99
VPI	3	37

Table: A short history of PLIs [1].

# What is Ruby?

## Definition

Ruby — a programmer's best friend

- Pure object-oriented scripting language
- Clean and thoughtful syntax
  - Easy to understand and *maintain*
  - Makes programming fun again!
- Highly portable
  - Works on UNIX, Mac, Windows, DOS, BeOS, etc.
- Open source software (GPL/artistic)
  - <http://ruby-lang.org>

# Why use Ruby?

- Friendly and helpful community
- Unlimited length integers
- Abstract data structures
- Regular expressions
- Iterators and closures<sup>2</sup>
- Portable multithreading<sup>3</sup>
- Easy to extend using C

---

<sup>2</sup>Functional programming.

<sup>3</sup>Yes, it even works in DOS!

# What is Ruby-VPI?

## Definition

Ruby-VPI — Ruby interface to Verilog<sup>a</sup> VPI

<sup>a</sup>IEEE Std. 1364-2005 and 1364-2001

- Enjoy VPI without the burden of C
  - `#ifdef` for simulators and platforms
  - Integers limited to 32 or 64 bits
  - Lack of abstract data structures
- Works with any VPI simulator
  - Synopsys VCS
  - Mentor Modelsim
  - Icarus Verilog
  - GPL CVer
- Open source software (GPL)

# How is Ruby-VPI used?

- 1 Start with a design you want to verify
- 2 Generate a test using automated test generator
- 3 Identify your expectations about the design
- 4 Add your expectations to the specification
- 5 Make design satisfy the specification
- 6 Run the test to verify the design

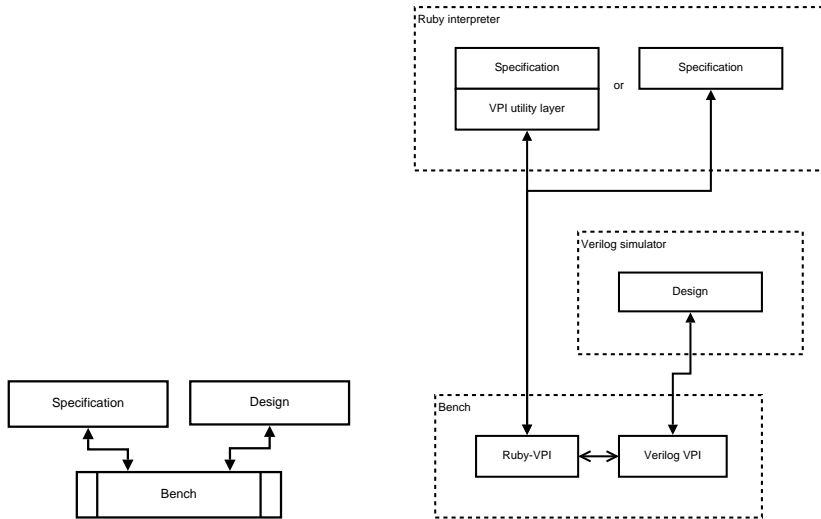
# Incremental development

- 1 Choose *one* expectation to verify
  - 2 Add expectation to specification
  - 3 Make design<sup>4</sup> satisfy expectation
  - 4 Verify design against specification
- Repeat until finished

---

<sup>4</sup>A Ruby prototype of the design can also be used.

# How does Ruby-VPI work?



# Verilog and Ruby interaction

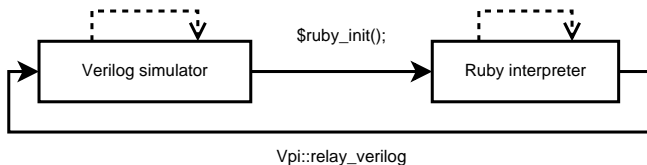


Figure: Initializing Ruby from Verilog.

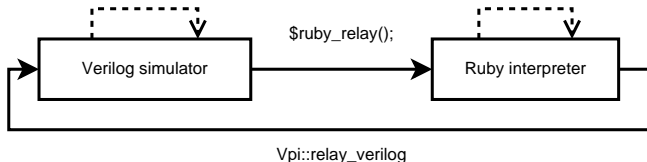


Figure: Relaying control between Ruby and Verilog.

# Ruby's view of VPI

## Definition

Handle — reference to a Verilog object under simulation  
e.g. registers, wires, modules, if-statements, etc.

- `Vpi` module encapsulates all access to VPI
- VPI types, structures, and constants become capitalized
- Method calls give access to a handle's VPI properties
- Handles are instances of `SWIG::TYPE_p_unsigned_int` class

# Getting a handle on things

## Obtain a handle using an absolute path.

```
a = vpi_handle_by_name("a", nil)
b = vpi_handle_by_name("a.b", nil)
c = vpi_handle_by_name("a.b.c", nil)
```

## Obtain a handle using a relative path.

```
a = vpi_handle_by_name("a", nil)
b = vpi_handle_by_name("b", a)
c = vpi_handle_by_name("c", b)
```

# Working with handles

## Obtain a handle's parent module.

- `parent = vpi_handle(VpiModule, handle)`
- `parent = handle.module`

## Iterate over a handle's nets.

- `handle[VpiNet].each {|net| ...}`
- `handle.each(VpiNet) {|net| ...}`
- `handle.each_net {|net| ...}`

## Obtain the absolute path to a handle.

- `path = vpi_get_str(VpiFullName, handle)`
- `path = handle.fullName`

# Working with logic values

## Read a handle's logic value as an integer.

- `value = handle.get_value(VpiIntVal)`
- `value = handle.vpiIntVal`
- `value = handle.intVal`

## Write an integer to a handle's logic value.

- `handle.put_value(15, VpiIntVal)`
- `handle.vpiIntVal = 15`
- `handle.intVal = 15`

`VpiBinStrVal`, `VpiOctStrVal`, `VpiDecStrVal`, `VpiHexStrVal`,  
`VpiStringVal`, `VpiScalarVal`, `VpiIntVal`, `VpiRealVal`, `VpiTimeVal`,  
`VpiVectorVal`, `VpiStrengthVal`



IEEE Std. 1364-2005:

*IEEE Standard Verilog Hardware Description Language.*

[http://ieeexplore.ieee.org/xpl/standardstoc.jsp?  
isnumber=33945](http://ieeexplore.ieee.org/xpl/standardstoc.jsp?isnumber=33945)



Brian Schröder:

*Ruby Course — an immersive programming course.*

<http://ruby.brian-schroeder.de/course/>



Ruby community:

*Ruby Programming Language.*

<http://ruby-lang.org>