### CONTENTS

1 Introduction ................................................. 3
   1.1 What is cocotb? ....................................... 3
   1.2 How is cocotb different? .............................. 3
   1.3 How does cocotb work? ............................... 4
   1.4 Contributors ......................................... 4

2 Quickstart Guide ............................................. 5
   2.1 Installing cocotb ..................................... 5
   2.2 Running your first Example ......................... 7
   2.3 Using cocotb ........................................ 7

3 Build options and Environment Variables .................... 11
   3.1 Make System .......................................... 11
   3.2 Environment Variables .............................. 12

4 Coroutines .................................................. 15
   4.1 Async functions .................................... 17

5 Triggers ...................................................... 19
   5.1 Simulator Triggers ................................. 19
   5.2 Python Triggers .................................... 21

6 Testbench Tools .............................................. 25
   6.1 Logging ............................................... 25
   6.2 Buses ............................................... 26
   6.3 Driving Buses ....................................... 26
   6.4 Monitoring Buses ................................... 27
   6.5 Tracking testbench errors ......................... 28

7 Library Reference ............................................ 29
   7.1 Test Results ......................................... 29
   7.2 Writing and Generating tests ...................... 30
   7.3 Interacting with the Simulator .................... 32
   7.4 Testbench Structure ................................ 37
   7.5 Utilities ............................................ 42
   7.6 Simulation Object Handles ....................... 45
   7.7 Implemented Testbench Structures ............... 49
   7.8 Miscellaneous ...................................... 53
   7.9 Developer-focused ................................. 53

8 C Code Library Reference .................................. 57
Contents:
1.1 What is cocotb?

cocotb is a COroutine based COSimulation TestBench environment for verifying VHDL/Verilog RTL using Python. cocotb is completely free, open source (under the BSD License) and hosted on GitHub.

cocotb requires a simulator to simulate the RTL. Simulators that have been tested and known to work with cocotb:

Linux Platforms
- Icarus Verilog
- GHDL
- Aldec Riviera-PRO
- Synopsys VCS
- Cadence Incisive and Xcelium
- Mentor ModelSim (DE and SE)
- Verilator

Windows Platform
- Icarus Verilog
- Aldec Riviera-PRO
- Mentor ModelSim (DE and SE)

A (possibly older) version of cocotb can be used live in a web-browser using EDA Playground.

1.2 How is cocotb different?

cocotb encourages the same philosophy of design re-use and randomized testing as UVM, however is implemented in Python rather than SystemVerilog.

With cocotb, VHDL/Verilog/SystemVerilog are normally only used for the design itself, not the testbench.

cocotb has built-in support for integrating with the Jenkins continuous integration system.

cocotb was specifically designed to lower the overhead of creating a test.

cocotb automatically discovers tests so that no additional step is required to add a test to a regression.
All verification is done using Python which has various advantages over using SystemVerilog or VHDL for verification:

- Writing Python is **fast** - it’s a very productive language
- It’s **easy** to interface to other languages from Python
- Python has a huge library of existing code to **re-use** like packet generation libraries.
- Python is **interpreted**. Tests can be edited and re-run them without having to recompile the design or exit the simulator GUI.
- Python is **popular** - far more engineers know Python than SystemVerilog or VHDL

### 1.3 How does cocotb work?

#### 1.3.1 Overview

A typical cocotb testbench requires no additional RTL code. The Design Under Test (DUT) is instantiated as the toplevel in the simulator without any wrapper code. cocotb drives stimulus onto the inputs to the DUT (or further down the hierarchy) and monitors the outputs directly from Python.

A test is simply a Python function. At any given time either the simulator is advancing time or the Python code is executing. The *yield* keyword is used to indicate when to pass control of execution back to the simulator. A test can spawn multiple coroutines, allowing for independent flows of execution.

### 1.4 Contributors

cocotb was developed by Potential Ventures with the support of Solarflare Communications Ltd and contributions from Gordon McGregor and Finn Grimwood (see contributors for the full list of contributions).

We also have a list of talks and papers, libraries and examples at our Wiki page Further Resources. Feel free to add links to cocotb-related content that we are still missing!
2.1 Installing cocotb

2.1.1 Pre-requisites

Cocotb has the following requirements:

- Python 2.7, Python 3.5+ (recommended)
- Python-dev packages
- GCC 4.8.1+ and associated development packages
- GNU Make
- A Verilog or VHDL simulator, depending on your RTL source code

2.1.2 Installation via PIP

New in version 1.2.

Cocotb can be installed by running

```bash
pip3 install cocotb
```

or

```bash
pip install cocotb
```

For user local installation follow the pip User Guide.

To install the development version of cocotb:

```bash
git clone https://github.com/cocotb/cocotb
pip install -e ./cocotb
```
2.1.3 Native Linux Installation

The following instructions will allow building of the cocotb libraries for use with a 64-bit native simulator.
If a 32-bit simulator is being used then additional steps are needed, please see our Wiki.

**Debian/Ubuntu-based**

```
sudo apt-get install git make gcc g++ swig python-dev
```

**Red Hat-based**

```
sudo yum install gcc gcc-c++ libstdc++-devel swig python-devel
```

2.1.4 Windows Installation

Download the MinGW installer from [https://osdn.net/projects/mingw/releases/](https://osdn.net/projects/mingw/releases/).

Run the GUI installer and specify a directory you would like the environment installed in. The installer will retrieve a list of possible packages, when this is done press “Continue”. The MinGW Installation Manager is then launched.

The following packages need selecting by checking the tick box and selecting “Mark for installation”

```
Basic Installation
-- mingw-developer-tools
-- mingw32-base
-- mingw32-gcc-g++
-- msys-base
```

From the Installation menu then select “Apply Changes”, in the next dialog select “Apply”.

When installed a shell can be opened using the `msys.bat` file located under the `<install_dir>/msys/1.0/`.

Python can be downloaded from [https://www.python.org/downloads/windows/](https://www.python.org/downloads/windows/). Run the installer and download to your chosen location.

It is beneficial to add the path to Python to the Windows system PATH variable so it can be used easily from inside Msys.

Once inside the Msys shell commands as given here will work as expected.

2.1.5 macOS Packages

You need a few packages installed to get cocotb running on macOS. Installing a package manager really helps things out here.

**Brew** seems to be the most popular, so we’ll assume you have that installed.

```
brew install python icarus-verilog gtkwave
```
2.2 Running your first Example

Assuming you have installed the prerequisites as above, the following lines are all you need to run a first simulation with cocotb:

```
git clone https://github.com/cocotb/cocotb
cd cocotb/examples/endian_swapper/tests
make
```

Selecting a different simulator is as easy as:

```
make SIM=vcs
```

2.2.1 Running the same example as VHDL

The `endian_swapper` example includes both a VHDL and a Verilog RTL implementation. The cocotb testbench can execute against either implementation using VPI for Verilog and VHPI/FLI for VHDL. To run the test suite against the VHDL implementation use the following command (a VHPI or FLI capable simulator must be used):

```
make SIM=ghdl TOPLEVEL_LANG=vhdl
```

2.3 Using cocotb

A typical cocotb testbench requires no additional HDL code (though nothing prevents you from adding testbench helper code). The Design Under Test (DUT) is instantiated as the toplevel in the simulator without any wrapper code. Cocotb drives stimulus onto the inputs to the DUT and monitors the outputs directly from Python.

2.3.1 Creating a Makefile

To create a cocotb test we typically have to create a Makefile. Cocotb provides rules which make it easy to get started. We simply inform cocotb of the source files we need compiling, the toplevel entity to instantiate and the Python test script to load.

```
VERILOG_SOURCES = $(PWD)/submodule.sv $(PWD)/my_design.sv
# TOPLEVEL is the name of the toplevel module in your Verilog or VHDL file:
TOPLEVEL=my_design
# MODULE is the name of the Python test file:
MODULE=test_my_design
include $(shell cocotb-config --makefiles)/Makefile.inc
include $(shell cocotb-config --makefiles)/Makefile.sim
```

We would then create a file called `test_my_design.py` containing our tests.
2.3.2 Creating a test

The test is written in Python. Cocotb wraps your top level with the handle you pass it. In this documentation, and most of the examples in the project, that handle is `dut`, but you can pass your own preferred name in instead. The handle is used in all Python files referencing your RTL project. Assuming we have a toplevel port called `clk` we could create a test file containing the following:

```python
import cocotb
from cocotb.triggers import Timer

@cocotb.test()
def my_first_test(dut):
    """Try accessing the design.""
    dut._log.info("Running test!")
    for cycle in range(10):
        dut.clk = 0
        yield Timer(1, units='ns')
        dut.clk = 1
        yield Timer(1, units='ns')
    dut._log.info("Running test!")
```

This will drive a square wave clock onto the `clk` port of the toplevel.

2.3.3 Accessing the design

When cocotb initializes it finds the top-level instantiation in the simulator and creates a handle called `dut`. Top-level signals can be accessed using the “dot” notation used for accessing object attributes in Python. The same mechanism can be used to access signals inside the design.

```python
# Get a reference to the "clk" signal on the top-level
clk = dut.clk

# Get a reference to a register "count"
# in a sub-block "inst_sub_block"
count = dut.inst_sub_block.count
```

2.3.4 Assigning values to signals

Values can be assigned to signals using either the `value` property of a handle object or using direct assignment while traversing the hierarchy.

```python
# Get a reference to the "clk" signal and assign a value
clk = dut.clk
clk.value = 1

# Direct assignment through the hierarchy
dut.input_signal <= 12

# Assign a value to a memory deep in the hierarchy
dut.sub_block.memory.array[4] <= 2
```

The syntax `sig <= new_value` is a short form of `sig.value = new_value`. It not only resembles HDL syntax, but also has the same semantics: writes are not applied immediately, but delayed until the next write cycle. Use `sig.setimmediatevalue(new_val)` to set a new value immediately (see `setimmediatevalue()`).
2.3.5 Reading values from signals

Accessing the `value` property of a handle object will return a `BinaryValue` object. Any unresolved bits are preserved and can be accessed using the `binstr` attribute, or a resolved integer value can be accessed using the `integer` attribute.

```python
>>> # Read a value back from the DUT
>>> count = dut.counter.value
>>> print(count.binstr)
1X1010
>>> # Resolve the value to an integer (X or Z treated as 0)
>>> print(count.integer)
42
>>> # Show number of bits in a value
>>> print(count.n_bits)
6
```

We can also cast the signal handle directly to an integer:

```python
>>> print(int(dut.counter))
42
```

2.3.6 Parallel and sequential execution

A `yield` will run a function (that must be marked as a “coroutine”, see Coroutines) sequentially, i.e. wait for it to complete. If a coroutine should be run “in the background”, i.e. in parallel to other coroutines, the way to do this is to `fork()` it. The end of such a forked coroutine can be waited on by using `join()`.

The following example shows these in action:

```python
@cocotb.coroutine
def reset_dut(reset_n, duration):
    reset_n <= 0
    yield Timer(duration, units='ns')
    reset_n <= 1
    reset_n._log.debug("Reset complete")

@cocotb.test()
def parallel_example(dut):
    reset_n = dut.reset

    # This will call reset_dut sequentially
    # Execution will block until reset_dut has completed
    yield reset_dut(reset_n, 500)
    dut._log.debug("After reset")

    # Call reset_dut in parallel with the 250 ns timer
    reset_thread = cocotb.fork(reset_dut(reset_n, 500))

    yield Timer(250, units='ns')
    dut._log.debug("During reset (reset_n = %s)" % reset_n.value)

    # Wait for the other thread to complete
    yield reset_thread.join()
    dut._log.debug("After reset")
```
CHAPTER THREE

BUILD OPTIONS AND ENVIRONMENT VARIABLES

3.1 Make System

Makefiles are provided for a variety of simulators in `cocotb/share/makefiles/simulators`. The common Makefile `cocotb/share/makefiles/Makefile.sim` includes the appropriate simulator Makefile based on the contents of the `SIM` variable.

3.1.1 Make Targets

Makefiles define two targets, `regression` and `sim`, the default target is `sim`.

Both rules create a results file with the name taken from `COCOTB_RESULTS_FILE`, defaulting to `results.xml`. This file is a JUnit-compatible output file suitable for use with Jenkins. The `sim` targets unconditionally re-runs the simulator whereas the `regression` target only re-builds if any dependencies have changed.

3.1.2 Make Phases

Typically the makefiles provided with cocotb for various simulators use a separate `compile` and `run` target. This allows for a rapid re-running of a simulator if none of the RTL source files have changed and therefore the simulator does not need to recompile the RTL.

3.1.3 Make Variables

- **GUI**
  
  Set this to 1 to enable the GUI mode in the simulator (if supported).

- **SIM**
  
  Selects which simulator Makefile to use. Attempts to include a simulator specific makefile from `cocotb/share/makefiles/makefile.$(SIM)`

- **WAVES**
  
  Set this to 1 to enable wave traces dump for the Aldec Riviera-PRO and Mentor Graphics Questa simulators. To get wave traces in Icarus Verilog see Simulator Support.

- **VERILOG_SOURCES**
  
  A list of the Verilog source files to include.

- **VHDL_SOURCES**
  
  A list of the VHDL source files to include.

- **VHDL_SOURCES_<lib>**
  
  A list of the VHDL source files to include in the VHDL library `lib` (currently for the GHDL simulator only).
**COMPILE_ARGS**

Any arguments or flags to pass to the compile stage of the simulation.

**SIM_ARGS**

Any arguments or flags to pass to the execution of the compiled simulation.

**EXTRA_ARGS**

Passed to both the compile and execute phases of simulators with two rules, or passed to the single compile and run command for simulators which don’t have a distinct compilation stage.

**PLUSARGS**

“Plusargs” are options that are starting with a plus (+) sign. They are passed to the simulator and are also available within cocotb as `cocotb.plusargs`. In the simulator, they can be read by the Verilog/SystemVerilog system functions `$test$plusargs` and `$value$plusargs`.

The special plusargs `+ntb_random_seed` and `+seed`, if present, are evaluated to set the random seed value if `RANDOM_SEED` is not set. If both `+ntb_random_seed` and `+seed` are set, `+ntb_random_seed` is used.

**COCOTB_HDL_TIMEUNIT**

The default time unit that should be assumed for simulation when not specified by modules in the design. If this isn’t specified then it is assumed to be lns. Allowed values are 1, 10, and 100. Allowed units are s, ms, us, ns, ps, fs.

New in version 1.3.

**COCOTB_HDL_TIMEPRECISION**

The default time precision that should be assumed for simulation when not specified by modules in the design. If this isn’t specified then it is assumed to be lps. Allowed values are 1, 10, and 100. Allowed units are s, ms, us, ns, ps, fs.

New in version 1.3.

**CUSTOM_COMPILE_DEPS**

Use to add additional dependencies to the compilation target; useful for defining additional rules to run pre-compilation or if the compilation phase depends on files other than the RTL sources listed in `VERILOG_SOURCES` or `VHDL_SOURCES`.

**CUSTOM_SIM_DEPS**

Use to add additional dependencies to the simulation target.

**COCOTB_NVC_TRACE**

Set this to 1 to enable display of VHPI traces when using the NVC VHDL simulator.

**SIM_BUILD**

Use to define a scratch directory for use by the simulator. The path is relative to the Makefile location. If not provided, the default scratch directory is `sim_build`.

### 3.2 Environment Variables

**TOPLEVEL**

Use this to indicate the instance in the hierarchy to use as the DUT. If this isn’t defined then the first root instance is used.

**RANDOM_SEED**

Seed the Python random module to recreate a previous test stimulus. At the beginning of every test a message is displayed with the seed used for that execution:
To recreate the same stimuli use the following:

```
made RANDOM_SEED=1377424946
```

See also: PLUSARGS

**COCOTB_ANSI_OUTPUT**
Use this to override the default behavior of annotating cocotb output with ANSI color codes if the output is a terminal (`isatty()`).

- `COCOTB_ANSI_OUTPUT=1` forces output to be ANSI regardless of the type of `stdout`
- `COCOTB_ANSI_OUTPUT=0` suppresses the ANSI output in the log messages

**COCOTB_REduced_LOG_FMT**
If defined, log lines displayed in the terminal will be shorter. It will print only time, message type (INFO, WARNING, ERROR, ...) and the log message itself.

**MODULE**
The name of the module(s) to search for test functions. Multiple modules can be specified using a comma-separated list.

**TESTCASE**
The name of the test function(s) to run. If this variable is not defined cocotb discovers and executes all functions decorated with the `cocotb.test` decorator in the supplied `MODULE` list.

Multiple test functions can be specified using a comma-separated list.

**COCOTB_RESULTS_FILE**
The file name where xUnit XML tests results are stored. If not provided, the default is `results.xml`.

New in version 1.3.

### 3.2.1 Additional Environment Variables

**COCOTB_ATTACH**
In order to give yourself time to attach a debugger to the simulator process before it starts to run, you can set the environment variable `COCOTB_ATTACH` to a pause time value in seconds. If set, cocotb will print the process ID (PID) to attach to and wait the specified time before actually letting the simulator run.

**COCOTB_ENABLE_PROFILING**
Enable performance analysis of the Python portion of cocotb. When set, a file `test_profile.pstat` will be written which contains statistics about the cumulative time spent in the functions.

From this, a callgraph diagram can be generated with `gprof2dot` and `graphviz`. See the `profile` Make target in the `endian_swapper` example on how to set this up.

**COCOTB_HOOKS**
A comma-separated list of modules that should be executed before the first test. You can also use the `cocotb.hook` decorator to mark a function to be run before test code.

**COCOTB_LOG_LEVEL**
The default logging level to use. This is set to INFO unless overridden. Valid values are DEBUG, INFO, WARNING, ERROR, CRITICAL.

**COCOTB_RESOLVE_X**
 Defines how to resolve bits with a value of X, Z, U or W when being converted to integer. Valid settings are:
VALUE_ERROR raise a ValueError exception
ZEROs resolve to 0
ONES resolve to 1
RANDOM randomly resolve to a 0 or a 1
Set to VALUE_ERROR by default.

COCOTB_SCHEDULER_DEBUG
Enable additional log output of the coroutine scheduler.

COVERAGE
Enable to report Python coverage data. For some simulators, this will also report HDL coverage.
This needs the coverage Python module to be installed.

MEMCHECK
HTTP port to use for debugging Python’s memory usage. When set to e.g. 8088, data will be presented at http://localhost:8088.
This needs the cherrypy and dowser Python modules installed.

COCOTB_PY_DIR
Path to the directory containing the cocotb Python package in the cocotb subdirectory. You don’t normally need to modify this.

COCOTB_SHARE_DIR
Path to the directory containing the cocotb Makefiles and simulator libraries in the subdirectories lib, include, and makefiles. You don’t normally need to modify this.
Testbenches built using cocotb use coroutines. While the coroutine is executing the simulation is paused. The coroutine uses the `yield` keyword to pass control of execution back to the simulator and simulation time can advance again.

Typically coroutines yield a `Trigger` object which indicates to the simulator some event which will cause the coroutine to be woken when it occurs. For example:

```python
@cocotb.coroutine
def wait_10ns():
    cocotb.log.info("About to wait for 10 ns")
    yield Timer(10, units='ns')
    cocotb.log.info("Simulation time has advanced by 10 ns")
```

Coroutines may also yield other coroutines:

```python
@cocotb.coroutine
def wait_100ns():
    for i in range(10):
        yield wait_10ns()
```

Coroutines can return a value, so that they can be used by other coroutines. Before Python 3.3, this requires a `ReturnValue` to be raised.

```python
@cocotb.coroutine
def get_signal(clk, signal):
    yield RisingEdge(clk)
    raise ReturnValue(signal.value)

@cocotb.coroutine
def get_signal_python_33(clk, signal):
    # newer versions of Python can use return normally
    yield RisingEdge(clk)
    return signal.value

@cocotb.coroutine
def check_signal_changes(dut):
    first = yield get_signal(dut.clk, dut.signal)
    second = yield get_signal(dut.clk, dut.signal)
    if first == second:
        raise TestFailure("Signal did not change")
```

Coroutines may also yield a list of triggers and coroutines to indicate that execution should resume if any of them fires:

```python
@cocotb.coroutine
def packet_with_timeout(monitor, timeout):
    (continues on next page)"
The trigger that caused execution to resume is passed back to the coroutine, allowing them to distinguish which trigger fired:

```python
@cocotb.coroutine
def packet_with_timeout(monitor, timeout):
    """Wait for a packet but time out if nothing arrives""
    tout_trigger = Timer(timeout, units='ns')
    result = yield [tout_trigger, RisingEdge(dut.ready)]
    if result is tout_trigger:
        raise TestFailure("Timed out waiting for packet")
```

Coroutines can be forked for parallel operation within a function of that code and the forked code.

```python
@cocotb.test()
def test_act_during_reset(dut):
    """While reset is active, toggle signals""
    tb = uart_tb(dut)
    # "Clock" is a built in class for toggling a clock signal
    cocotb.fork(Clock(dut.clk, 1, units='ns').start())
    # reset_dut is a function -
    # part of the user-generated "uart_tb" class
    cocotb.fork(tb.reset_dut(dut.rstn, 20))

    yield Timer(10, units='ns')
    print("Reset is still active: %d" % dut.rstn)
    yield Timer(15, units='ns')
    print("Reset has gone inactive: %d" % dut.rstn)
```

Coroutines can be joined to end parallel operation within a function.

```python
@cocotb.test()
def test_count_edge_cycles(dut, period=1, clocks=6):
    cocotb.fork(Clock(dut.clk, period, units='ns').start())
    yield RisingEdge(dut.clk)
    timer = Timer(period + 10)
    task = cocotb.fork(count_edges_cycles(dut.clk, clocks))
    count = 0
    expect = clocks - 1

    while True:
        result = yield [timer, task.join()]
        if count > expect:
            raise TestFailure("Task didn't complete in expected time")
        if result is timer:
            dut._log.info("Count %d: Task still running" % count)
            count += 1
        else:
            break
```

Coroutines can be killed before they complete, forcing their completion before they’d naturally end.
@cocotb.test()
def test_different_clocks(dut):
    clk_1mhz = Clock(dut.clk, 1.0, units='us')
    clk_250mhz = Clock(dut.clk, 4.0, units='ns')

    clk_gen = cocotb.fork(clk_1mhz.start())
    start_time_ns = get_sim_time(units='ns')
    yield Timer(1, units='ns')
    yield RisingEdge(dut.clk)
    edge_time_ns = get_sim_time(units='ns')
    # NOTE: isclose is a Python 3.5+ feature
    if not isclose(edge_time_ns, start_time_ns + 1000.0):
        raise TestFailure("Expected a period of 1 us")
    clk_gen.kill()

    clk_gen = cocotb.fork(clk_250mhz.start())
    start_time_ns = get_sim_time(units='ns')
    yield Timer(1, units='ns')
    yield RisingEdge(dut.clk)
    edge_time_ns = get_sim_time(units='ns')
    # NOTE: isclose is a Python 3.5+ feature
    if not isclose(edge_time_ns, start_time_ns + 4.0):
        raise TestFailure("Expected a period of 4 ns")

4.1 Async functions

Python 3.5 introduces **async** functions, which provide an alternative syntax. For example:

```python
@cocotb.coroutine
async def wait_10ns():
    cocotb.log.info("About to wait for 10 ns")
    await Timer(10, units='ns')
    cocotb.log.info("Simulation time has advanced by 10 ns")
```

To wait on a trigger or a nested coroutine, these use **await** instead of **yield**. Provided they are decorated with @cocotb.coroutine, **async def** functions using **await** and regular functions using **yield** can be used interchangeably - the appropriate keyword to use is determined by which type of function it appears in, not by the sub-coroutine being called.

**Note:** It is not legal to **await** a list of triggers as can be done in **yield**-based coroutine with **yield** [trig1, trig2]. Use **await** First(trig1, trig2) instead.
4.1.1 Async generators

In Python 3.6, a `yield` statement within an `async` function has a new meaning (rather than being a `SyntaxError`) which matches the typical meaning of `yield` within regular Python code. It can be used to create a special type of generator function that can be iterated with `async for`:

```python
async def ten_samples_of(clk, signal):
    for i in range(10):
        await RisingEdge(clk)
        yield signal.value  # this means "send back to the for loop"

@cocotb.test()
async def test_samples_are_even(dut):
    async for sample in ten_samples_of(dut.clk, dut.signal):
        assert sample % 2 == 0
```

More details on this type of generator can be found in PEP 525.
Triggers are used to indicate when the cocotb scheduler should resume coroutine execution. To use a trigger, a coroutine should `await` or `yield` it. This will cause execution of the current coroutine to pause. When the trigger fires, execution of the paused coroutine will resume:

```python
@cocotb.coroutine
def coro():
    print("Some time before the edge")
    yield RisingEdge(clk)
    print("Immediately after the edge")
```

Or using the syntax in Python 3.5 onwards:

```python
@cocotb.coroutine
async def coro():
    print("Some time before the edge")
    await RisingEdge(clk)
    print("Immediately after the edge")
```

### 5.1 Simulator Triggers

#### 5.1.1 Signals

```python
class cocotb.triggers.Edge(signal)
    Fires on any value change of `signal`.

class cocotb.triggers.RisingEdge(signal)
    Fires on the rising edge of `signal`, on a transition from 0 to 1.

class cocotb.triggers.FallingEdge(signal)
    Fires on the falling edge of `signal`, on a transition from 1 to 0.

class cocotb.triggers.ClockCycles(signal, num_cycles, rising=True)
    Fires after `num_cycles` transitions of `signal` from 0 to 1.
```

**Parameters**

- `signal` – The signal to monitor.
- `num_cycles` *(int)* – The number of cycles to count.
- `rising` *(bool, optional)* – If True, the default, count rising edges. Otherwise, count falling edges.
5.1.2 Timing

class cocotb.triggers.Timer(time_ps, units=None)
Fires after the specified simulation time period has elapsed.

Parameters

- **time_ps** (**numbers.Real** or **decimal.Decimal**) – The time value. Note that despite the name this is not actually in picoseconds but depends on the **units** argument.
- **units** (**str** or **None**, optional) – One of **None**. 'fs', 'ps', 'ns', 'us', 'ms', 'sec'. When no **units** is given (**None**) the timestep is determined by the simulator.

Examples

```python
>>> yield Timer(100, units='ps')
```

The time can also be a float:

```python
>>> yield Timer(100e-9, units='sec')
```

which is particularly convenient when working with frequencies:

```python
>>> freq = 10e6  # 10 MHz
>>> yield Timer(1 / freq, units='sec')
```

Other builtin exact numeric types can be used too:

```python
>>> from fractions import Fraction
>>> yield Timer(Fraction(1, 10), units='ns')
```

```python
>>> from decimal import Decimal
>>> yield Timer(Decimal('100e-9'), units='sec')
```

These are most useful when using computed durations while avoiding floating point inaccuracies.

See also:

- get_sim_steps()

class cocotb.triggers.ReadOnly
Fires when the current simulation timestep moves to the read-only phase.

The read-only phase is entered when the current timestep no longer has any further delta steps. This will be a point where all the signal values are stable as there are no more RTL events scheduled for the timestep. The simulator will not allow scheduling of more events in this timestep. Useful for monitors which need to wait for all processes to execute (both RTL and cocotb) to ensure sampled signal values are final.

class cocotb.triggersReadWrite
Fires when the read-write portion of the sim cycles is reached.

class cocotb.triggers.NextTimeStep
Fires when the next time step is started.
5.2 Python Triggers

```python
class cocotb.triggers.Combine(*triggers)
    Fires when all of triggers have fired.

    Like most triggers, this simply returns itself.

class cocotb.triggers.First(*triggers)
    Fires when the first trigger in triggers fires.

    Returns the result of the trigger that fired.

As a shorthand, `t = yield [a, b]` can be used instead of `t = yield First(a, b)`. Note that this shorthand is not available when using `await`.
```

**Note:** The event loop is single threaded, so while events may be simultaneous in simulation time, they can never be simultaneous in real time. For this reason, the value of `t_ret` is `t1` in the following example is implementation-defined, and will vary by simulator:

```python
t1 = Timer(10, units='ps')
t2 = Timer(10, units='ps')
t_ret = yield First(t1, t2)
```

```python
class cocotb.triggers.Join(coroutine)
    Fires when a `fork()`ed coroutine completes.

    The result of blocking on the trigger can be used to get the coroutine result:
```

```python
@cocotb.coroutine()
def coro_inner():
    yield Timer(1, units='ns')
    raise ReturnValue("Hello world")

task = cocotb.fork(coro_inner())
result = yield Join(task)
assert result == "Hello world"
```

Or using the syntax in Python 3.5 onwards:

```python
@cocotb.coroutine()
async def coro_inner():
    await Timer(1, units='ns')
    return "Hello world"

task = cocotb.fork(coro_inner())
result = await Join(task)
assert result == "Hello world"
```

If the coroutine threw an exception, the `await` or `yield` will re-raise it.

```python
property retval
    The return value of the joined coroutine.
```

**Note:** Typically there is no need to use this attribute - the following code samples are equivalent:
foraked = cocotb.fork(mycoro())
j = Join(foraked)
yield j
result = j.retval

foraked = cocotb.fork(mycoro())
result = yield Join(foraked)

### 5.2.1 Synchronization

These are not Triggers themselves, but contain methods that can be used as triggers. These are used to synchronize coroutines with each other.

**class cocotb.triggers.Event(name='')**

Event to permit synchronization between two coroutines.

Yielding `wait()` from one coroutine will block the coroutine until `set()` is called somewhere else.

**set(data=None)**

Wake up all coroutines blocked on this event.

**wait()**

Get a trigger which fires when another coroutine sets the event.

If the event has already been set, the trigger will fire immediately.

To reset the event (and enable the use of `wait` again), `clear()` should be called.

**clear()**

Clear this event that has fired.

Subsequent calls to `wait()` will block until `set()` is called again.

**class cocotb.triggers.Lock(name='')**

Lock primitive (not re-entrant).

This should be used as:

```python
yield lock.acquire()
try:
    # do some stuff
finally:
    lock.release()
```

locked = None

True if the lock is held.

**acquire()**

Produce a trigger which fires when the lock is acquired.

**release()**

Release the lock.

**cocotb.triggers.with_timeout()**

Waits on triggers, throws an exception if it waits longer than the given time.

Usage:
Parameters

- **trigger** (*cocotb_waitable*) – A single object that could be right of a `yield` (or `await` in Python 3) expression in cocotb.
- **timeout_time** (*numbers.Real or decimal.Decimal*) – Time duration.
- **timeout_unit** (*str or None, optional*) – Units of duration, accepts any values that `Timer` does.

**Returns** First trigger that completed if timeout did not occur.

**Raises** *SimTimeoutError* – If timeout occurs.

New in version 1.3.
CHAPTER SIX

TESTBENCH TOOLS

6.1 Logging

Cocotb extends the Python logging library. Each DUT, monitor, driver, and scoreboard (as well as any other function using the coroutine decorator) implements its own logging object, and each can be set to its own logging level. Within a DUT, each hierarchical object can also have individual logging levels set.

When logging HDL objects, beware that \_log is the preferred way to use logging. This helps minimize the change of name collisions with an HDL log component with the Python logging functionality.

Log printing levels can also be set on a per-object basis.

```python
class EndianSwapperTB(object):
    def __init__(self, dut, debug=False):
        self.dut = dut
        self.stream_in = AvalonSTDriver(dut, "stream_in", dut.clk)
        self.stream_in_recovered = AvalonSTMonitor(dut, "stream_in", dut.clk,
                                                  callback=self.model)

        # Set verbosity on our various interfaces
        level = logging.DEBUG if debug else logging.WARNING
        self.stream_in.log.setLevel(level)
        self.stream_in_recovered.log.setLevel(level)
        self.dut.reset_n._log.setLevel(logging.DEBUG)
```

And when the logging is actually called

```python
class AvalonSTPkts(BusMonitor):
    ...
    @coroutine
    def _monitor_recv(self):
        ...
        self.log.info("Received a packet of \$d bytes" % len(pkt))

class Scoreboard(object):
    ...
    def add_interface(self):
        ...
        self.log.info("Created with reorder_depth \$d" % reorder_depth)

class EndianSwapTB(object):
    ...
    @cocotb.coroutine
```

(continues on next page)
def reset():
    self.dut._log.debug("Resetting DUT")

will display as something like

<table>
<thead>
<tr>
<th>Time</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00ns</td>
<td>INFO cocotb.scoreboard.endian_swapper_sv scoreboard.</td>
</tr>
<tr>
<td>-py:177</td>
<td>in add_interface Created with reorder_depth 0</td>
</tr>
<tr>
<td>0.00ns</td>
<td>DEBUG cocotb.endian_swapper_sv .._endian_swapper.</td>
</tr>
<tr>
<td>-py:106</td>
<td>in reset Resetting DUT</td>
</tr>
<tr>
<td>160000000000000.00ns</td>
<td>INFO cocotb.endian_swapper_sv.stream_out avalon.</td>
</tr>
<tr>
<td>-py:151</td>
<td>in _monitor_recv Received a packet of 125 bytes</td>
</tr>
</tbody>
</table>

6.2 Buses

Buses are simply defined as collection of signals. The Bus class will automatically bundle any group of signals together that are named similar to dut.<bus_name><separator><signal_name>. For instance,

dut.stream_in_valid
dut.stream_in_data

have a bus name of stream_in, a separator of _, and signal names of valid and data. A list of signal names, or a dictionary mapping attribute names to signal names is also passed into the Bus class. Buses can have values driven onto them, be captured (returning a dictionary), or sampled and stored into a similar object.

stream_in_bus = Bus(dut, "stream_in", ["valid", "data"]) # '//' is the default separator

6.3 Driving Buses

Examples and specific bus implementation bus drivers (AMBA, Avalon, XGMII, and others) exist in the Driver class enabling a test to append transactions to perform the serialization of transactions onto a physical interface. Here is an example using the Avalon bus driver in the endian_swapper example:

class EndianSwapperTB(object):
    def __init__(self, dut, debug=False):
        self.dut = dut
        self.stream_in = AvalonSTDriver(dut, "stream_in", dut.clk)
    def run_test(dut, data_in=None, configCoroutine=None, idle_inserter=None, backpressure_inserter=None):
        cocotb.fork(Clock(dut.clk, 5000).start())
        tb = EndianSwapperTB(dut)
        yield tb.reset()
        dut.stream_out_ready <= 1
        if idle_inserter is not None:
            tb.stream_in.set_valid_generator(idle_inserter())
# Send in the packets
for transaction in data_in():
    yield tb.stream_in.send(transaction)

## 6.4 Monitoring Buses

For our testbenches to actually be useful, we have to monitor some of these buses, and not just drive them. That’s where the Monitor class comes in, with pre-built monitors for Avalon and XGMII buses. The Monitor class is a base class which you are expected to derive for your particular purpose. You must create a `monitor_recv()` function which is responsible for determining 1) at what points in simulation to call the `recv()` function, and 2) what transaction values to pass to be stored in the monitors receiving queue. Monitors are good for both outputs of the DUT for verification, and for the inputs of the DUT, to drive a test model of the DUT to be compared to the actual DUT. For this purpose, input monitors will often have a callback function passed that is a model. This model will often generate expected transactions, which are then compared using the Scoreboard class.

```python
# ============================================================================
class BitMonitor(Monitor):
    """Observes single input or output of DUT."""
    def __init__(self, name, signal, clock, callback=None, event=None):
        self.name = name
        self.signal = signal
        self.clock = clock
        Monitor.__init__(self, callback, event)

@coroutine
    def monitor_recv(self):
        clkedge = RisingEdge(self.clock)

        while True:
            # Capture signal at rising edge of clock
            yield clkedge
            vec = self.signal.value
            self._recv(vec)

    # ============================================================================
def input_gen():
        """Generator for input data applied by BitDriver"""
        while True:
            yield random.randint(1,5), random.randint(1,5)

    # ============================================================================
class DFF_TB(object):
        def __init__(self, dut, init_val):
            self.dut = dut

            # Create input driver and output monitor
            self.input_drv = BitDriver(dut.d, dut.c, input_gen())
            self.output_mon = BitMonitor("output", dut.q, dut.c)

            # Create a scoreboard on the outputs
            self.expected_output = [ init_val ]
```

(continues on next page)
Reconstruct the input transactions from the pins and send them to our 'model':

```python
self.input_mon = BitMonitor("input", dut.d, dut.c, callback=self.model)
```

```python
def model(self, transaction):
    """Model the DUT based on the input transaction."""
    if not self.stopped:
        self.expected_output.append(transaction)
```

### 6.5 Tracking testbench errors

The `Scoreboard` class is used to compare the actual outputs to expected outputs. Monitors are added to the scoreboard for the actual outputs, and the expected outputs can be either a simple list, or a function that provides a transaction. Here is some code from the `dff` example, similar to above with the scoreboard added.

```python
class DFF_TB(object):
    def __init__(self, dut, init_val):
        self.dut = dut

        # Create input driver and output monitor
        self.input_drv = BitDriver(dut.d, dut.c, input_gen())
        self.output_mon = BitMonitor("output", dut.q, dut.c)

        # Create a scoreboard on the outputs
        self.expected_output = [ init_val ]
        self.scoreboard = Scoreboard(dut)
        self.scoreboard.add_interface(self.output_mon, self.expected_output)

        # Reconstruct the input transactions from the pins
        # and send them to our 'model'
        self.input_mon = BitMonitor("input", dut.d, dut.c,callback=self.model)
```
7.1 Test Results

The exceptions in this module can be raised at any point by any code and will terminate the test.

```python
    cocotb.result.raise_error(obj, msg)
    # Create a TestError exception and raise it after printing a traceback.
    # Deprecated since version 1.3: Use raise TestError(msg) instead of this function. A stacktrace will be
    # printed by cocotb automatically if the exception is unhandled.
    Parameters
    - obj – Object with a log method.
    - msg (str) – The log message.
```

```python
    cocotb.result.create_error(obj, msg)
    # Like raise_error(), but return the exception rather than raise it, simply to avoid too many levels of nested
    # try/except blocks.
    # Deprecated since version 1.3: Use TestError(msg) directly instead of this function.
    Parameters
    - obj – Object with a log method.
    - msg (str) – The log message.
```

exception cocotb.result.ReturnValue(retval)
    Helper exception needed for Python versions prior to 3.3.

exception cocotb.result.TestComplete(*args, **kwargs)
    Exception showing that the test was completed. Sub-exceptions detail the exit status.

exception cocotb.result.ExternalException(exception)
    Exception thrown by cocotb.external functions.

exception cocotb.result.TestError(*args, **kwargs)
    Exception showing that the test was completed with severity Error.

exception cocotb.result.TestFailure(*args, **kwargs)
    Exception showing that the test was completed with severity Failure.

exception cocotb.result.TestSuccess(*args, **kwargs)
    Exception showing that the test was completed successfully.

exception cocotb.result.SimFailure(*args, **kwargs)
    Exception showing that the simulator exited unsuccessfully.
**exception cocotb.result.SimTimeoutError**
Exception for when a timeout, in terms of simulation time, occurs.

## 7.2 Writing and Generating tests

**class cocotb.test** *(f, timeout_time=None, timeout_unit=None, expect_fail=False, expect_error=False, skip=False, stage=None)*
Decorator to mark a function as a test.

All tests are coroutines. The test decorator provides some common reporting etc., a test timeout and allows us to mark tests as expected failures.

Used as `@cocotb.test(...)`.

**Parameters**

- **timeout_time (int, optional)** – Value representing simulation timeout.
  
  New in version 1.3.

- **timeout_unit (str, optional)** – Unit of timeout value, see `Timer` for more info.
  
  New in version 1.3.

- **expect_fail (bool, optional)** – Don’t mark the result as a failure if the test fails.

- **expect_error (bool or exception type or tuple of exception types, optional)** – If True, consider this test passing if it raises *any* `Exception`, and failing if it does not. If given an exception type or tuple of exception types, catching only a listed exception type is considered passing. This is primarily for cocotb internal regression use for when a simulator error is expected.

  Users are encouraged to use the following idiom instead:

  ```python
  @cocotb.test()
  def my_test(dut):
      try:
          yield thing_that_should_fail()
      except ExceptionIExpect:
          pass
      else:
          assert False, "Exception did not occur"
  ```

  Changed in version 1.3: Specific exception types can be expected

- **skip (bool, optional)** – Don’t execute this test as part of the regression.

- **stage (int, optional)** – Order tests logically into stages, where multiple tests can share a stage.

**class cocotb.coroutine (func)**
Decorator class that allows us to provide common coroutine mechanisms:

- log methods will log to `cocotb.coroutine.name`.

- `join()` method returns an event which will fire when the coroutine exits.

  Used as `@cocotb.coroutine`.

**class cocotb.external (func)**
Decorator to apply to an external function to enable calling from cocotb.
This turns a normal function that isn’t a coroutine into a blocking coroutine. Currently, this creates a new execution thread for each function that is called. Scope for this to be streamlined to a queue in future.

**class cocotb.function(func)**

Decorator class that allows a function to block.

This allows a coroutine that consumes simulation time to be called by a thread started with `cocotb.external`; in other words, to internally block while externally appear to yield.

**class cocotb.hook(f)**

Decorator to mark a function as a hook for cocotb.

Used as `@cocotb.hook()`.

All hooks are run at the beginning of a cocotb test suite, prior to any test code being run.

**class cocotb.regression.TestFactory(test_function, *args, **kwargs)**

Factory to automatically generate tests.

**Parameters**

- **test_function** – The function that executes a test. Must take `dut` as the first argument.
- ***args** – Remaining arguments are passed directly to the test function. Note that these arguments are not varied. An argument that varies with each test must be a keyword argument to the test function.
- ****kwargs** – Remaining keyword arguments are passed directly to the test function. Note that these arguments are not varied. An argument that varies with each test must be a keyword argument to the test function.

Assuming we have a common test function that will run a test. This test function will take keyword arguments (for example generators for each of the input interfaces) and generate tests that call the supplied function.

This Factory allows us to generate sets of tests based on the different permutations of the possible arguments to the test function.

For example if we have a module that takes backpressure and idles and have some packet generation routines `gen_a` and `gen_b`:

```python
>>> tf = TestFactory(test_function=run_test)
>>> tf.add_option(name='data_in', optionlist=[gen_a, gen_b])
>>> tf.add_option('backpressure', [None, random_backpressure])
>>> tf.add_option('idles', [None, random_idles])
>>> tf.generate_tests()
```

We would get the following tests:

- `gen_a` with no backpressure and no idles
- `gen_a` with no backpressure and random_idles
- `gen_a` with random_backpressure and no idles
- `gen_a` with random_backpressure and random_idles
- `gen_b` with no backpressure and no idles
- `gen_b` with no backpressure and random_idles
- `gen_b` with random_backpressure and no idles
- `gen_b` with random_backpressure and random_idles

7.2. Writing and Generating tests
The tests are appended to the calling module for auto-discovery.

Tests are simply named `test_function_N`. The docstring for the test (hence the test description) includes the name and description of each generator.

```python
add_option(name, optionlist)
```
Add a named option to the test.

Parameters

- `name` (`str`) – Name of the option. Passed to test as a keyword argument.
- `optionlist` (`list`) – A list of possible options for this test knob.

```python
generate_tests(prefix='', postfix='')
```
Generate an exhaustive set of tests using the cartesian product of the possible keyword arguments.

The generated tests are appended to the namespace of the calling module.

Parameters

- `prefix` (`str`) – Text string to append to start of `test_function` name when naming generated test cases. This allows reuse of a single `test_function` with multiple `TestFactories` without name clashes.
- `postfix` (`str`) – Text string to append to end of `test_function` name when naming generated test cases. This allows reuse of a single `test_function` with multiple `TestFactories` without name clashes.

### 7.3 Interacting with the Simulator

```python
class cocotb.binary.BinaryRepresentation
```

UNSIGNED = 0
Unsigned format

SIGNED_MAGNITUDE = 1
Sign and magnitude format

TWOS_COMPLEMENT = 2
Two’s complement format

```python
class cocotb.binary.BinaryValue(value=None, n_bits=None, bigEndian=True, binaryRepresentation=0, bits=None)
```
Representation of values in binary format.

The underlying value can be set or accessed using these aliasing attributes:

- `BinaryValue.integer` is an integer
- `BinaryValue.signed_integer` is a signed integer
- `BinaryValue.binstr` is a string of 01xXzZ
- `BinaryValue.buff` is a binary buffer of bytes
- `BinaryValue.value` is an integer deprecated

For example:
Parameters

- **value** *(str or int or long, optional)* – Value to assign to the bus.
- **n_bits** *(int, optional)* – Number of bits to use for the underlying binary representation.
- **bigEndian** *(bool, optional)* – Interpret the binary as big-endian when converting to/from a string buffer.
- **binaryRepresentation** *(BinaryRepresentation)* – The representation of the binary value (one of **UNSIGNED**, **SIGNED_MAGNITUDE**, **TWOS_COMPLEMENT**). Defaults to unsigned representation.
- **bits** *(int, optional)* – Deprecated: Compatibility wrapper for **n_bits**.

**assign** *(value)*

Decides how best to assign the value to the vector.

We possibly try to be a bit too clever here by first of all trying to assign the raw string as a **BinaryValue. binstr**, however if the string contains any characters that aren’t 0, 1, X or Z then we interpret the string as a binary buffer.

**Parameters** **value** *(str or int or long)* – The value to assign.

**get_value** *

Return the integer representation of the underlying vector.

**get_value_signed** *

Return the signed integer representation of the underlying vector.

**property is_resolvable**

Does the value contain any X’s? Inquiring minds want to know.

**property value**

Integer access to the value. **deprecated**

**property integer**

The integer representation of the underlying vector.

**property signed_integer**

The signed integer representation of the underlying vector.

**get_buff** *

Attribute **buff** represents the value as a binary string buffer.

```python
>>> "0100000100101111".buff == "A/
True
```

**property buff**

Access to the value as a buffer.

**get_binstr** *

Attribute **binstr** is the binary representation stored as a string of 1 and 0.
**property binstr**
Access to the binary string.

**property n_bits**
Access to the number of bits of the binary value.

class cocotb.bus.Bus(entity, name, signals, optional_signals=[], bus_separator='_', array_idx=None)
Wraps up a collection of signals.
Assumes we have a set of signals/nets named entity.<bus_name><separator><signal>.
For example a bus stream_in with signals valid and data is assumed to be named dut.
stream_in_valid and dut.stream_in_data (with the default separator '_').

**Todo:** Support for struct/record ports where signals are member names.

### Parameters

- **entity** *(SimHandle)* – *SimHandle* instance to the entity containing the bus.
- **name** *(str)* – Name of the bus. None for a nameless bus, e.g. bus-signals in an interface or a modport (untested on struct/record, but could work here as well).
- **signals** *(list or dict)* – In the case of an object (passed to drive()/capture()) that has the same attribute names as the signal names of the bus, the signals argument can be a list of those names. When the object has different attribute names, the signals argument should be a dict that maps bus attribute names to object signal names.
- **optional_signals** *(list or dict, optional)* – Signals that don’t have to be present on the interface. See the signals argument above for details.
- **bus_separator** *(str, optional)* – Character(s) to use as separator between bus name and signal name. Defaults to ‘_’.
- **array_idx** *(int or None, optional)* – Optional index when signal is an array.

**drive**(obj, strict=False)
Drives values onto the bus.

**Parameters**

- **obj** – Object with attribute names that match the bus signals.
- **strict** *(bool, optional)* – Check that all signals are being assigned.

**Raises** *AttributeError* – If not all signals have been assigned when strict=True.

**capture**()
Capture the values from the bus, returning an object representing the capture.

**Returns** A dictionary that supports access by attribute, where each attribute corresponds to each signal’s value.

**Return type** *dict*

**Raises** *RuntimeError* – If signal not present in bus, or attempt to modify a bus capture.

**sample**(obj, strict=False)
Sample the values from the bus, assigning them to obj.

**Parameters**
• **obj** – Object with attribute names that match the bus signals.

• **strict** *(bool, optional)* – Check that all signals being sampled are present in **obj**.

**Raises** AttributeError – If attribute is missing in **obj** when strict=True.

```python
class cocotb.clock.Clock(signal, period, units=None)
```

Simple 50:50 duty cycle clock driver.

Instances of this class should call its `start()` method and `fork()` the result. This will create a clocking thread that drives the signal at the desired period/frequency.

Example:

```python
c = Clock(dut.clk, 10, 'ns')
cocotb.fork(c.start())
```

**Parameters**

• **signal** – The clock pin/signal to be driven.

• **period** *(int)* – The clock period. Must convert to an even number of timesteps.

• **units** *(str, optional)* – One of None, 'fs', 'ps', 'ns', 'us', 'ms', 'sec'.

When no **units** is given (None) the timestep is determined by the simulator.

If you need more features like a phase shift and an asymmetric duty cycle, it is simple to create your own clock generator (that you then `fork()`):

```python
@cocotb.coroutine
def custom_clock():
    # pre-construct triggers for performance
    high_time = Timer(high_delay, units="ns")
    low_time = Timer(low_delay, units="ns")
    yield Timer(initial_delay, units="ns")
    while True:
        dut.clk <= 1
        yield high_time
        dut.clk <= 0
        yield low_time
```

If you also want to change the timing during simulation, use this slightly more inefficient example instead where the Timers inside the while loop are created with current delay values:

```python
@cocotb.coroutine
def custom_clock():
    while True:
        dut.clk <= 1
        yield Timer(high_delay, units="ns")
        dut.clk <= 0
        yield Timer(low_delay, units="ns")
```

```
high_delay = low_delay = 100
cocotb.fork(custom_clock())
yield Timer(1000, units="ns")
high_delay = low_delay = 10  # change the clock speed
yield Timer(1000, units="ns")
```

**cocotb.fork (coroutine)**

Add a new coroutine.
Just a wrapper around self.schedule which provides some debug and useful error messages in the event of common gotchas.

cocotb.decorators.RunningCoroutine.join(self)
Return a trigger that will fire when the wrapped coroutine exits.

cocotb.decorators.RunningCoroutine.kill(self)
Kill a coroutine.

7.3.1 Triggers

See *Simulator Triggers* for a list of sub-classes. Below are the internal classes used within cocotb.

class cocotb.triggers.Trigger
Base class to derive from.

    abstract prime(callback)
    Set a callback to be invoked when the trigger fires.

    The callback will be invoked with a single argument, self.

    Sub-classes must override this, but should end by calling the base class method.

    Do not call this directly within coroutines, it is intended to be used only by the scheduler.

unprime()
Remove the callback, and perform cleanup if necessary.

    After being un-primed, a Trigger may be re-primed again in the future. Calling unprime multiple times is
    allowed, subsequent calls should be a no-op.

    Sub-classes may override this, but should end by calling the base class method.

    Do not call this directly within coroutines, it is intended to be used only by the scheduler.

class cocotb.triggers.GPITrigger
Base Trigger class for GPI triggers.
Consumes simulation time.

unprime()
Disable a primed trigger, can be re-primed.

class cocotb.triggers.Waitable
Compatibility layer that emulates collections.abc.Awaitable.
This converts a _wait abstract method into a suitable __await__ on supporting Python versions (>=3.3).

    _wait
    Should be implemented by the sub-class. Called by yield self to convert the waitable object into a coroutine.

    ReturnValue can be used here.
7.4 Testbench Structure

7.4.1 Driver

class cocotb.drivers.Driver

Class defining the standard interface for a driver within a testbench.

The driver is responsible for serializing transactions onto the physical pins of the interface. This may consume simulation time.

Constructor for a driver instance.

kill()

Kill the coroutine sending stuff.

append(transaction, callback=None, event=None, **kwargs)

Queue up a transaction to be sent over the bus.

Parameters

• transaction (any) – The transaction to be sent.
• callback (callable, optional) – Optional function to be called when the transaction has been sent.
• event (optional) – Event to be set when the transaction has been sent.
• **kwargs – Any additional arguments used in child class’ _driver_send method.

clear()

Clear any queued transactions without sending them onto the bus.

send

Blocking send call (hence must be “yielded” rather than called).

Sends the transaction over the bus.

Parameters

• transaction (any) – The transaction to be sent.
• sync (bool, optional) – Synchronize the transfer by waiting for a rising edge.
• **kwargs (dict) – Additional arguments used in child class’ _driver_send method.

__driver_send__(transaction, sync=True, **kwargs)

Actual implementation of the send.

Sub-classes should override this method to implement the actual send() routine.

Parameters

• transaction (any) – The transaction to be sent.
• sync (bool, optional) – Synchronize the transfer by waiting for a rising edge.
• **kwargs – Additional arguments if required for protocol implemented in a sub-class.

__send__

Send coroutine.

Parameters
• **transaction** *(any)* – The transaction to be sent.
• **callback** *(callable, optional)* – Optional function to be called when the transaction has been sent.
• **event** *(optional)* – Event to be set when the transaction has been sent.
• **sync** *(bool, optional)* – Synchronize the transfer by waiting for a rising edge.
• ****kwargs** – Any additional arguments used in child class’ `_driver_send` method.

class cocotb.drivers.BitDriver(signal, clk, generator=None)
   Bases: object

Drives a signal onto a single bit.
Useful for exercising ready/valid flags.

**start**(generator=None)
Start generating data.

**Parameters**
**generator** *(generator, optional)* – Generator yielding data. The generator should yield tuples *(on, off)* with the number of cycles to be on, followed by the number of cycles to be off. Typically the generator should go on forever.

Example:

```
bit_driver.start(((1, i % 5) for i in itertools.count()))
```

**stop**()
Stop generating data.

class cocotb.drivers.BusDriver(entity, name, clock, **kwargs)
   Bases: cocotb.drivers.Driver

Wrapper around common functionality for buses which have:
• a list of `_signals` (class attribute)
• a list of `_optional_signals` (class attribute)
• a clock
• a name
• an entity

**Parameters**
• **entity** *(SimHandle)* – A handle to the simulator entity.
• **name** *(str or None)* – Name of this bus. `None` for a nameless bus, e.g. bus-signals in an interface or a modport. (untested on struct/record, but could work here as well).
• **clock** *(SimHandle)* – A handle to the clock associated with this bus.
• ****kwargs** *(dict)* – Keyword arguments forwarded to `cocotb.Bus`, see docs for that class for more information.

Constructor for a driver instance.

**_driver_send**
Implementation for BusDriver.

**Parameters**
• transaction – The transaction to send.
• sync (bool, optional) – Synchronize the transfer by waiting for a rising edge.

_wait_for_signal
This method will return when the specified signal has hit logic 1. The state will be in the ReadOnly phase so sim will need to move to NextTimeStep before registering more callbacks can occur.

_wait_for_nsignal
This method will return when the specified signal has hit logic 0. The state will be in the ReadOnly phase so sim will need to move to NextTimeStep before registering more callbacks can occur.

class cocotb.drivers.ValidatedBusDriver (entity, name, clock, **kwargs)
Bases: cocotb.drivers.BusDriver

Same as a BusDriver except we support an optional generator to control which cycles are valid.

Parameters
• entity (SimHandle) – A handle to the simulator entity.
• name (str) – Name of this bus.
• clock (SimHandle) – A handle to the clock associated with this bus.
• valid_generator (generator, optional) – a generator that yields tuples of (valid, invalid) cycles to insert.

Constructor for a driver instance.

_next_valids ()
Optionally insert invalid cycles every N cycles.

The generator should yield tuples with the number of cycles to be on followed by the number of cycles to be off. The on cycles should be non-zero, we skip invalid generator entries.

set_valid_generator (valid_generator=None)
Set a new valid generator for this bus.

7.4.2 Monitor
class cocotb.monitors.Monitor (callback=None, event=None)
Base class for Monitor objects.

Monitors are passive ‘listening’ objects that monitor pins going in or out of a DUT. This class should not be used directly, but should be sub-classed and the internal _monitor_recv method should be overridden and decorated as a coroutine. This _monitor_recv method should capture some behavior of the pins, form a transaction, and pass this transaction to the internal _recv method. The _monitor_recv method is added to the cocotb scheduler during the __init__ phase, so it should not be yielded anywhere.

The primary use of a Monitor is as an interface for a Scoreboard.

Parameters
• callback (callable) – Callback to be called with each recovered transaction as the argument. If the callback isn’t used, received transactions will be placed on a queue and the event used to notify any consumers.
• event (cocotb.triggers.Event) – Event that will be called when a transaction is received through the internal _recv method. Event.data is set to the received transaction.

wait_for_recv (timeout=None)
With timeout, wait () for transaction to arrive on monitor and return its data.
Parameters **timeout** – The timeout value for **Timer**. Defaults to **None**.

**Returns** Data of received transaction.

```python
_monitor_recv
Actual implementation of the receiver.
Sub-classes should override this method to implement the actual receive routine and call _recv with the recovered transaction.

_recv(transaction)
Common handling of a received transaction.
```

class cocotb.monitors.BusMonitor(entity, name, clock, reset=None, reset_n=None, callback=None, event=None, bus_separator='_', array_idx=None)

**Bases:** cocotb.monitors.Monitor

Wrapper providing common functionality for monitoring buses.

**property in_reset**
Boolean flag showing whether the bus is in reset state or not.

### 7.4.3 Scoreboard

Common scoreboarding capability.

class cocotb.scoreboard.Scoreboard(dut, reorder_depth=0, fail_immediately=True)

**Bases:** object

Generic scoreboarding class.

We can add interfaces by providing a monitor and an expected output queue.

The expected output can either be a function which provides a transaction or a simple list containing the expected output.

**Todo:** Statistics for end-of-test summary etc.

**Parameters**

- **dut (SimHandle)** – Handle to the DUT.
- **reorder_depth (int, optional)** – Consider up to reorder_depth elements of the expected result list as passing matches. Default is 0, meaning only the first element in the expected result list is considered for a passing match.
- **fail_immediately (bool, optional)** – Raise **TestFailure** immediately when something is wrong instead of just recording an error. Default is **True**.

**property result**

Determine the test result, do we have any pending data remaining?

**Returns** If not all expected output was received or error were recorded during the test.

**Return type** TestFailure

```python
compare(got, exp, log, strict_type=True)
Common function for comparing two transactions.
Can be re-implemented by a sub-class.
```
Parameters

- **got** – The received transaction.
- **exp** – The expected transaction.
- **log** – The logger for reporting messages.
- **strict_type** *(bool, optional)* – Require transaction type to match exactly if True, otherwise compare its string representation.

Raises **TestFailure** – If received transaction differed from expected transaction when `fail_immediately` is True. If `strict_type` is True, also the transaction type must match.

```
add_interface (monitor, expected_output, compare_fn=None, reorder_depth=0, strict_type=True)
```

Add an interface to be scoreboarded.

Provides a function which the monitor will callback with received transactions.

Simply check against the expected output.

Parameters

- **monitor** – The monitor object.
- **expected_output** – Queue of expected outputs.
- **compare_fn** *(callable, optional)* – Function doing the actual comparison.
- **reorder_depth** *(int, optional)* – Consider up to `reorder_depth` elements of the expected result list as passing matches. Default is 0, meaning only the first element in the expected result list is considered for a passing match.
- **strict_type** *(bool, optional)* – Require transaction type to match exactly if True, otherwise compare its string representation.

Raises **TypeError** – If no monitor is on the interface or `compare_fn` is not a callable function.

### 7.4.4 Clock

```
class cocotb.clock.Clock (signal, period, units=None)
```

Simple 50:50 duty cycle clock driver.

Instances of this class should call its `start()` method and `fork()` the result. This will create a clocking thread that drives the signal at the desired period/frequency.

Example:

```
c = Clock(dut.clk, 10, 'ns')
cocotb.fork(c.start())
```

Parameters

- **signal** – The clock pin/signal to be driven.
- **period** *(int)* – The clock period. Must convert to an even number of timesteps.
- **units** *(str, optional)* – One of None, 'fs', 'ps', 'ns', 'us', 'ms', 'sec'. When no `units` is given (None) the timestep is determined by the simulator.

If you need more features like a phase shift and an asymmetric duty cycle, it is simple to create your own clock generator (that you then `fork()`):
@cocotb.coroutine
def custom_clock():
    # pre-construct triggers for performance
    high_time = Timer(high_delay, units="ns")
    low_time = Timer(low_delay, units="ns")
    yield Timer(initial_delay, units="ns")
    while True:
        dut.clk <= 1
        yield high_time
        dut.clk <= 0
        yield low_time

If you also want to change the timing during simulation, use this slightly more inefficient example instead where the Timer s inside the while loop are created with current delay values:

@cocotb.coroutine
def custom_clock():
    while True:
        dut.clk <= 1
        yield Timer(high_delay, units="ns")
        dut.clk <= 0
        yield Timer(low_delay, units="ns")

    high_delay = low_delay = 100
    cocotb.fork(custom_clock())
    yield Timer(1000, units="ns")
    high_delay = low_delay = 10  # change the clock speed
    yield Timer(1000, units="ns")

start
Clocking coroutine. Start driving your clock by fork()ing a call to this.

Parameters

- **cycles** *(int, optional)* – Cycle the clock *cycles* number of times, or if *None* then cycle the clock forever. Note: 0 is not the same as *None*, as 0 will cycle no times.

- **start_high** *(bool, optional)* – Whether to start the clock with a 1 for the first half of the period. Default is True.

New in version 1.3.

7.5 Utilities

cocotb.plusargs = {}
A dictionary of “plusargs” handed to the simulation.

cocotb.utils.get_sim_time *(units=None)*
Retrieves the simulation time from the simulator.

Parameters **units** *(str or None, optional)* – String specifying the units of the result (one of *None*, 'fs', 'ps', 'ns', 'us', 'ms', 'sec'). *None* will return the raw simulation time.

Returns The simulation time in the specified units.

cocotb.utils.get_time_from_sim_steps *(steps, units)*
Calculates simulation time in the specified *units* from the *steps* based on the simulator precision.
Parameters

- **steps (int)** – Number of simulation steps.
- **units (str)** – String specifying the units of the result (one of 'fs', 'ps', 'ns', 'us', 'ms', 'sec').

**Returns** The simulation time in the specified units.

cocotb.utils.get_sim_steps (time, units=None)
Calculates the number of simulation time steps for a given amount of `time`.

Parameters

- **time (numbers.Real or decimal.Decimal)** – The value to convert to simulation time steps.
- **units (str or None, optional)** – String specifying the units of the result (one of None, 'fs', 'ps', 'ns', 'us', 'ms', 'sec'). None means time is already in simulation time steps.

**Returns** The number of simulation time steps.

**Return type** int

**Raises** ValueError – If given `time` cannot be represented by simulator precision.

cocotb.utils.pack (ctypes_obj)
Convert a ctypes structure into a Python string.

Parameters **ctypes_obj (ctypes.Structure)** – The ctypes structure to convert to a string.

**Returns** New Python string containing the bytes from memory holding ctypes_obj.

cocotb.utils.unpack (ctypes_obj, string, bytes=None)
Unpack a Python string into a ctypes structure.

If the length of `string` is not the correct size for the memory footprint of the ctypes structure then the bytes keyword argument must be used.

Parameters

- **ctypes_obj (ctypes.Structure)** – The ctypes structure to pack into.
- **string (str)** – String to copy over the ctypes_obj memory space.
- **bytes (int, optional)** – Number of bytes to copy. Defaults to None, meaning the length of `string` is used.

**Raises**
- **ValueError** – If length of `string` and size of ctypes_obj are not equal.
- **MemoryError** – If `bytes` is longer than size of ctypes_obj.

cocotb.utils.hexdump (x)
Hexdump a buffer.

Parameters **x** – Object that supports conversion via the str built-in.

**Returns** A string containing the hexdump.
Example

```python
>>> print(hexdump('this somewhat long string'))
 0000 74 68 69 73 20 73 6F 6D 65 77 68 61 74 20 6C 6F this somewhat lo
 0010 6E 67 20 73 74 72 69 6E 67 ng string
```

cocotb.utils.hexdiffs(x, y)

Return a diff string showing differences between two binary strings.

**Parameters**

- `x` – Object that supports conversion via the `str` built-in.
- `y` – Object that supports conversion via the `str` built-in.

Example

```python
>>> print(hexdiffs(0, 1))
 0000 30 0
 0000 31 1
>>> print(hexdiffs('a', 'b'))
 0000 61 a
 0000 62 b
>>> print(hexdiffs('this short thing', 'this also short'))
 0000 746869732073686F 7274207468696E67 this short thing
 0000 7468697320616C73 6F 2073686F7274 this also short
```

cocotb.utils.ParametrizedSingleton(*args, **kwargs)

A metaclass that allows class construction to reuse an existing instance.

We use this so that `RisingEdge(sig)` and `Join(coroutine)` always return the same instance, rather than creating new copies.

cocotb.utils.reject_remaining_kwargs(name, kwars)

Helper function to emulate Python 3 keyword-only arguments.

Use as:

```python
def func(x1, **kwargs):
    a = kwargs.pop('a', 1)
    b = kwargs.pop('b', 2)
    reject_remaining_kwargs('func', kwargs)
    ...
```

To emulate the Python 3 syntax:

```python
def func(x1, *, a=1, b=2):
    ...
```

cocotb.utils.lazy_property(fget)

A property that is executed the first time, then cached forever.

It does this by replacing itself on the instance, which works because unlike `@property` it does not define `__set__`. This should be used for expensive members of objects that are not always used.
 cocotb.utils.want_color_output()
    Return True if colored output is possible/requested and not running in GUI.

    Colored output can be explicitly requested by setting `COCOTB_ANSI_OUTPUT` to 1.

cocotb.utils.remove_traceback_frames(tb_or_exc, frame_names)
    Strip leading frames from a traceback

    Parameters

        • `tb_or_exc` (Union[traceback, BaseException, exc_info]) – Object to strip frames from. If an exception is passed, creates a copy of the exception with a new shorter traceback. If a tuple from `sys.exc_info` is passed, returns the same tuple with the traceback shortened

        • `frame_names` (List[str]) – Names of the frames to strip, which must be present.

7.6 Simulation Object Handles

```
class cocotb.handle.SimHandleBase(handle, path)
    Bases: object

    Base class for all simulation objects.

    We maintain a handle which we can use for GPI calls.

    Parameters

        • `handle` (int) – The GPI handle to the simulator object.

        • `path` (str) – Path to this handle, None if root.

class cocotb.handle.RegionObject(handle, path)
    Bases: cocotb.handle.SimHandleBase

    A region object, such as a scope or namespace.

    Region objects don’t have values, they are effectively scopes or namespaces.

    Parameters

        • `handle` (int) – The GPI handle to the simulator object.

        • `path` (str) – Path to this handle, None if root.
```

class cocotb.handle.HierarchyObject (handle, path)
    Bases: cocotb.handle.RegionObject

Hierarchy objects are namespace/scope objects.

Parameters

* handle (int) – The GPI handle to the simulator object.
* path (str) – Path to this handle, None if root.

class cocotb.handle.HierarchyArrayObject (handle, path)
    Bases: cocotb.handle.RegionObject

Hierarchy Arrays are containers of Hierarchy Objects.

Parameters

* handle (int) – The GPI handle to the simulator object.
* path (str) – Path to this handle, None if root.

class cocotb.handle.NonHierarchyObject (handle, path)
    Bases: cocotb.handle.SimHandleBase

Common base class for all non-hierarchy objects.

Parameters

* handle (int) – The GPI handle to the simulator object.
* path (str) – Path to this handle, None if root.

property value
A reference to the value

class cocotb.handle.ConstantObject (handle, path, handle_type)
    Bases: cocotb.handle.NonHierarchyObject

An object which has a value that can be read, but not set.

We can also cache the value since it is fixed at elaboration time and won’t change within a simulation.

Parameters

* handle (int) – The GPI handle to the simulator object.
* path (str) – Path to this handle, None if root.
* handle_type – The type of the handle (simulator.INTEGER, simulator.ENUM, simulator.REAL, simulator.STRING).

class cocotb.handle.NonHierarchyIndexableObject (handle, path)
    Bases: cocotb.handle.NonHierarchyObject

A non-hierarchy indexable object.

Parameters

* handle (int) – The GPI handle to the simulator object.
* path (str) – Path to this handle, None if root.

class cocotb.handle.NonConstantObject (handle, path)
    Bases: cocotb.handle.NonHierarchyIndexableObject

A non-constant object

Parameters
• **handle** (*int*) – The GPI handle to the simulator object.

• **path** (*str*) – Path to this handle, *None* if root.

**drivers()**

An iterator for gathering all drivers for a signal.

**loads()**

An iterator for gathering all loads on a signal.

**class cocotb.handle.ModifiableObject(handle, path)**

Bases: `cocotb.handle.NonConstantObject`

Base class for simulator objects whose values can be modified.

**Parameters**

• **handle** (*int*) – The GPI handle to the simulator object.

• **path** (*str*) – Path to this handle, *None* if root.

**setimmediatevalue(value)**

Set the value of the underlying simulation object to *value*.

This operation will fail unless the handle refers to a modifiable object, e.g. net, signal or variable.

We determine the library call to make based on the type of the value because assigning integers less than 32 bits is faster.

**Parameters**

• **value** (*ctypes.Structure, cocotb.binary.BinaryValue, int, double*) – The value to drive onto the simulator object.

**Raises** **TypeError** – If target has an unsupported type for value assignment.

**class cocotb.handle.RealObject(handle, path)**

Bases: `cocotb.handle.ModifiableObject`

Specific object handle for Real signals and variables.

**Parameters**

• **handle** (*int*) – The GPI handle to the simulator object.

• **path** (*str*) – Path to this handle, *None* if root.

**setimmediatevalue(value)**

Set the value of the underlying simulation object to *value*.

This operation will fail unless the handle refers to a modifiable object, e.g. net, signal or variable.

**Parameters**

• **value** (*float*) – The value to drive onto the simulator object.

**Raises** **TypeError** – If target has an unsupported type for real value assignment.

**class cocotb.handle.EnumObject(handle, path)**

Bases: `cocotb.handle.ModifiableObject`

Specific object handle for enumeration signals and variables.

**Parameters**

• **handle** (*int*) – The GPI handle to the simulator object.

• **path** (*str*) – Path to this handle, *None* if root.
setimmediatevalue(value)
Set the value of the underlying simulation object to value.
This operation will fail unless the handle refers to a modifiable object, e.g. net, signal or variable.

Parameters value(int) – The value to drive onto the simulator object.

Raises TypeError – If target has an unsupported type for integer value assignment.

class cocotb.handle.IntegerObject(handle, path)
Bases: cocotb.handle.ModifiableObject
Specific object handle for Integer and Enum signals and variables.

Parameters
• handle(int) – The GPI handle to the simulator object.
• path(str) – Path to this handle, None if root.

setimmediatevalue(value)
Set the value of the underlying simulation object to value.
This operation will fail unless the handle refers to a modifiable object, e.g. net, signal or variable.

Parameters value(int) – The value to drive onto the simulator object.

Increases TypeError – If target has an unsupported type for integer value assignment.

class cocotb.handle.StringObject(handle, path)
Bases: cocotb.handle.ModifiableObject
Specific object handle for String variables.

Parameters
• handle(int) – The GPI handle to the simulator object.
• path(str) – Path to this handle, None if root.

setimmediatevalue(value)
Set the value of the underlying simulation object to value.
This operation will fail unless the handle refers to a modifiable object, e.g. net, signal or variable.

Parameters value(str) – The value to drive onto the simulator object.

Raises TypeError – If target has an unsupported type for string value assignment.

cocotb.handle.SimHandle(handle, path=None)
Factory function to create the correct type of SimHandle object.

Parameters
• handle(int) – The GPI handle to the simulator object.
• path(str) – Path to this handle, None if root.

Returns The SimHandle object.

Raises TestError – If no matching object for GPI type could be found.
7.7 Implemented Testbench Structures

7.7.1 Drivers

AMBA

Advanced Microcontroller Bus Architecture.

class cocotb.drivers.amba.AXI4LiteMaster(entity, name, clock, **kwargs)

AXI4-Lite Master.

TODO: Kill all pending transactions if reset is asserted.

Constructor for a driver instance.

write(address, value, byte_enable=0xf, address_latency=0, data_latency=0)

Write a value to an address.

Parameters

• address (int) – The address to write to.
• value (int) – The data value to write.
• byte_enable (int, optional) – Which bytes in value to actually write. Default is to write all bytes.
• address_latency (int, optional) – Delay before setting the address (in clock cycles). Default is no delay.
• data_latency (int, optional) – Delay before setting the data value (in clock cycles). Default is no delay.
• sync (bool, optional) – Wait for rising edge on clock initially. Defaults to True.

Returns The write response value.

Return type BinaryValue

Raises AXIProtocolError – If write response from AXI is not OKAY.

read(address, sync=True)

Read from an address.

Parameters

• address (int) – The address to read from.
• sync (bool, optional) – Wait for rising edge on clock initially. Defaults to True.

Returns The read data value.

Return type BinaryValue

Raises AXIProtocolError – If read response from AXI is not OKAY.

class cocotb.drivers.amba.AXI4Slave(entity, name, clock, memory, callback=None, event=None, big_endian=False, **kwargs)

AXI4 Slave

Monitors an internal memory and handles read and write requests.

Constructor for a driver instance.
**Avalon**

```python
class cocotb.drivers.avalon.AvalonMM(entity, name, clock, **kwargs)
    Bases: cocotb.drivers.BusDriver

   Avalon Memory Mapped Interface (Avalon-MM) Driver.
   Currently we only support the mode required to communicate with SF avalon_mapper which is a limited subset of all the signals.
   Blocking operation is all that is supported at the moment, and for the near future as well. Posted responses from a slave are not supported.
   Constructor for a driver instance.

class cocotb.drivers.avalon.AvalonMaster(entity, name, clock, **kwargs)

   Avalon Memory Mapped Interface (Avalon-MM) Master.
   Constructor for a driver instance.

   write(address, value)
   Issue a write to the given address with the specified value.
   Parameters
       • address (int) – The address to write to.
       • value (int) – The data value to write.

   Raises TestError – If master is read-only.

   read(address, sync=True)
   Issue a request to the bus and block until this comes back.
   Simulation time still progresses but syntactically it blocks.
   Parameters
       • address (int) – The address to read from.
       • sync (bool, optional) – Wait for rising edge on clock initially. Defaults to True.

   Returns The read data value.
   Return type BinaryValue

   Raises TestError – If master is write-only.
```

```python
class cocotb.drivers.avalon.AvalonMemory(entity, name, clock, readlatency_min=1, readlatency_max=1, memory=None, aval_properties={}, **kwargs)
     Bases: cocotb.drivers.BusDriver

   Emulate a memory, with back-door access.
   Constructor for a driver instance.

class cocotb.drivers.avalon.AvalonST(entity, name, clock, **kwargs)
    Bases: cocotb.drivers.ValidatedBusDriver

   Avalon Streaming Interface (Avalon-ST) Driver
   Constructor for a driver instance.

class cocotb.drivers.avalon.AvalonSTPkts(entity, name, clock, **kwargs)
    Bases: cocotb.drivers.ValidatedBusDriver

   Avalon Streaming Interface (Avalon-ST) Driver, packetized.
```
Constructor for a driver instance.

**OPB**

```python
class cocotb.drivers.opb.OPBMaster (entity, name, clock, **kwargs)
```

On-chip peripheral bus master.

Constructor for a driver instance.

```python
write (address, value, sync=True)
```

Issue a write to the given address with the specified value.

- **Parameters**
  - `address (int)` – The address to read from.
  - `value (int)` – The data value to write.
  - `sync (bool, optional)` – Wait for rising edge on clock initially. Defaults to True.

- **Raises** `OPBException` – If write took longer than 16 cycles.

```python
read (address, sync=True)
```

Issue a request to the bus and block until this comes back.

Simulation time still progresses but syntactically it blocks.

- **Parameters**
  - `address (int)` – The address to read from.
  - `sync (bool, optional)` – Wait for rising edge on clock initially. Defaults to True.

- **Returns** The read data value.

- **Return type** `BinaryValue`

- **Raises** `OPBException` – If read took longer than 16 cycles.

**XGMII**

```python
class cocotb.drivers.xgmii.XGMII (signal, clock, interleaved=True)
```

XGMII (10 Gigabit Media Independent Interface) driver.

- **Parameters**
  - `signal (SimHandle)` – The XGMII data bus.
  - `clock (SimHandle)` – The associated clock (assumed to be driven by another coroutine).
  - `interleaved (bool, optional)` – Whether control bits are interleaved with the data bytes or not.

- **If interleaved the bus is** byte0, byte0_control, byte1, byte1_control, …

- **Otherwise expect** byte0, byte1, …, byte0_control, byte1_control, …

```python
static layer1 (packet)
```

Take an Ethernet packet (as a string) and format as a layer 1 packet.

Pad to 64 bytes, prepend preamble and append 4-byte CRC on the end.
Parameters `packet (str)`: The Ethernet packet to format.

Returns The formatted layer 1 packet.

Return type `str`

`idle ()`: Helper function to set bus to IDLE state.

`terminate (index)`: Helper function to terminate from a provided lane index.

Parameters `index (int)`: The index to terminate.

## 7.7.2 Monitors

### Avalon

class `cocotb.monitors.avalon.AvalonST (entity, name, clock, **kwargs)`

Bases: `cocotb.monitors.BusMonitor`

Avalon-ST bus.

Non-packetized so each valid word is a separate transaction.

class `cocotb.monitors.avalon.AvalonSTPkts (entity, name, clock, **kwargs)`

Bases: `cocotb.monitors.BusMonitor`

Packetized Avalon-ST bus.

Parameters

- `name, clock (entity,)`: see BusMonitor
- `config (dict)`: bus configuration options
- `report_channel (bool)`: report channel with data, default is False. Setting to True on bus without channel signal will give an error

### XGMII

class `cocotb.monitors.xgmii.XGMII (signal, clock, interleaved=True, callback=None, event=None)`

Bases: `cocotb.monitors.Monitor`

XGMII (10 Gigabit Media Independent Interface) Monitor.

Assumes a single vector, either 4 or 8 bytes plus control bit for each byte.

If interleaved is `True` then the control bits are adjacent to the bytes.

Parameters

- `signal (SimHandle)`: The XGMII data bus.
- `clock (SimHandle)`: The associated clock (assumed to be driven by another coroutine).
- `interleaved (bool, optional)`: Whether control bits are interleaved with the data bytes or not.

If interleaved the bus is byte0, byte0_control, byte1, byte1_control, . . .

Otherwise expect byte0, byte1, . . . , byte0_control, byte1_control, . . .
7.8 Miscellaneous

7.8.1 Signal Tracer for WaveDrom

class cocotb.wavedrom.Wavedrom(obj)
   Base class for a WaveDrom compatible tracer.
   
   sample()
      Record a sample of the signal value at this point in time.
   
   clear()
      Delete all sampled data.
   
   get(add_clock=True)
      Return the samples as a list suitable for use with WaveDrom.

class cocotb.wavedrom.trace(*args, **kwargs)
   Context manager to enable tracing of signals.
   
   Arguments are an arbitrary number of signals or buses to trace. We also require a clock to sample on, passed in as a keyword argument.
   
   Usage:

   ```python
   with trace(sig1, sig2, a_bus, clk=clk) as waves:
      # Stuff happens, we trace it
      # Dump to JSON format compatible with WaveDrom
      j = waves.dumpj()
   ```

7.9 Developer-focused

7.9.1 The Scheduler

Note: The scheduler object should generally not be interacted with directly - the only part of it that a user will need is encapsulated in `fork()`, everything else works behind the scenes.

```python
 cocotb.scheduler = <cocotb.scheduler.Scheduler object>  
    The global scheduler instance.

 class cocotb.scheduler.Scheduler
    The main scheduler.
    
    Here we accept callbacks from the simulator and schedule the appropriate coroutines.
    
    A callback fires, causing the react method to be called, with the trigger that caused the callback as the first argument.
    
    We look up a list of coroutines to schedule (indexed by the trigger) and schedule them in turn.

    Attention: Implementors should not depend on the scheduling order!
```
Some additional management is required since coroutines can return a list of triggers, to be scheduled when any one of the triggers fires. To ensure we don’t receive spurious callbacks, we have to un-prime all the other triggers when any one fires.

Due to the simulator nuances and fun with delta delays we have the following modes:

**Normal mode**
- Callbacks cause coroutines to be scheduled
- Any pending writes are cached and do not happen immediately

**ReadOnly mode**
- Corresponds to `cbReadOnlySynch` (VPI) or `vhpiCbLastKnownDeltaCycle` (VHPI). In this state we are not allowed to perform writes.

**Write mode**
- Corresponds to `cbReadWriteSynch` (VPI) or `vhpiCbEndOfProcesses` (VHPI). In this mode we play back all the cached write updates.

We can legally transition from Normal to Write by registering a `ReadWrite` callback, however usually once a simulator has entered the ReadOnly phase of a given timestep then we must move to a new timestep before performing any writes. The mechanism for moving to a new timestep may not be consistent across simulators and therefore we provide an abstraction to assist with compatibility.

Unless a coroutine has explicitly requested to be scheduled in ReadOnly mode (for example wanting to sample the finally settled value after all delta delays) then it can reasonably be expected to be scheduled during “normal mode” i.e. where writes are permitted.

**react** *(trigger)*
Called when a trigger fires.

We ensure that we only start the event loop once, rather than letting it recurse.

**unschedule** *(coro)*
Unschedule a coroutine. Unprime any pending triggers

**queue** *(coroutine)*
Queue a coroutine for execution

**queue_function** *(coro)*
Queue a coroutine for execution and move the containing thread so that it does not block execution of the main thread any longer.

**run_in_executor** *(func, *args, **kwargs)*
Run the coroutine in a separate execution thread and return a yieldable object for the caller.

**add** *(coroutine)*
Add a new coroutine.

Just a wrapper around `self.schedule` which provides some debug and useful error messages in the event of common gotchas.

**add_test** *(test_coro)*
Called by the regression manager to queue the next test

**schedule** *(coroutine, trigger=None)*
Schedule a coroutine by calling the send method.

**Parameters**
- **coroutine** *(cocotb.decorators.coroutine)* – The coroutine to schedule.
• **trigger**(*cocotb.triggers.Trigger*) – The trigger that caused this coroutine to be scheduled.

**finish_scheduler**(*exc*)

Directly call into the regression manager and end test once we return the sim will close us so no cleanup is needed.

**cleanup()**

Clear up all our state.

Unprime all pending triggers and kill off any coroutines stop all externals.
Cocotb contains a library called GPI (in directory `cocotb/share/lib/gpi/`) written in C++ that is an abstraction layer for the VPI, VHPI, and FLI simulator interfaces.

The interaction between Python and GPI is via a Python extension module called `simulator` (in directory `cocotb/share/lib/simulator/`) which provides routines for traversing the hierarchy, getting/setting an object's value, registering callbacks etc.

### 8.1 API Documentation

#### 8.1.1 Class list

**Class FliEnumObjHdl**

```cpp
class FliEnumObjHdl : public FliValueObjHdl
```
Class FliImpl

class FliImpl : public GpiImplInterface

Native Check Create

Determine whether a simulation object is native to FLI and create a handle if it is

Get current simulation time

Get current simulation time
NB units depend on the simulation configuration

Find the root handle

Find the root handle using an optional name
Get a handle to the root simulator object. This is usually the toplevel.
If no name is provided, we return the first root instance.
If name is provided, we check the name against the available objects until we find a match. If no match is found we return NULL.

Class FliIntObjHdl

class FliIntObjHdl : public FliValueObjHdl

Class FliIterator

class FliIterator : public GpiIterator

Find the root handle

Find the root handle using an optional name
Get a handle to the root simulator object. This is usually the toplevel.
If no name is provided, we return the first root instance.
If name is provided, we check the name against the available objects until we find a match. If no match is found we return NULL.
Class FliLogicObjHdl

class FliLogicObjHdl: public FliValueObjHdl

Class FliNextPhaseCbHdl

class FliNextPhaseCbHdl: public FliSimPhaseCbHdl

Class FliObj

class FliObj
    Subclassed by FliObjHdl, FliSignalObjHdl

Class FliObjHdl

class FliObjHdl: public GpiObjHdl, public FliObj

Class FliProcessCbHdl

class FliProcessCbHdl: public virtual GpiCbHdl
    Subclassed by FliShutdownCbHdl, FliSignalCbHdl, FliSimPhaseCbHdl, FliStartupCbHdl, FliTimedCbHdl

    cleanup callback

    Called while unwinding after a GPI callback
    We keep the process but desensitize it
    NB: need a way to determine if should leave it sensitized . . .

Class FliReadOnlyCbHdl

class FliReadOnlyCbHdl: public FliSimPhaseCbHdl

Class FliReadWriteCbHdl

class FliReadWriteCbHdl: public FliSimPhaseCbHdl

Class FliRealObjHdl

class FliRealObjHdl: public FliValueObjHdl
Class FliShutdownCbHdl

class FliShutdownCbHdl : public FliProcessCbHdl

    cleanup callback

    Called while unwinding after a GPI callback
    We keep the process but desensitize it
    NB: need a way to determine if should leave it sensitized . . .

Class FliSignalCbHdl

class FliSignalCbHdl : public FliProcessCbHdl, public GpiValueCbHdl

    cleanup callback

    Called while unwinding after a GPI callback
    We keep the process but desensitize it
    NB: need a way to determine if should leave it sensitized . . .

Class FliSignalObjHdl

class FliSignalObjHdl : public GpiSignalObjHdl, public FliObj
    Subclassed by FliValueObjHdl

Class FliSimPhaseCbHdl

class FliSimPhaseCbHdl : public FliProcessCbHdl
    Subclassed by FliNextPhaseCbHdl, FliReadOnlyCbHdl, FliReadWriteCbHdl

    cleanup callback

    Called while unwinding after a GPI callback
    We keep the process but desensitize it
    NB: need a way to determine if should leave it sensitized . . .

Class FliStartupCbHdl

class FliStartupCbHdl : public FliProcessCbHdl
cleanup callback

Called while unwinding after a GPI callback
We keep the process but desensitize it
NB: need a way to determine if should leave it sensitized…

Class FliStringObjHdl

class FliStringObjHdl : public FliValueObjHdl

Class FliTimedCbHdl

class FliTimedCbHdl : public FliProcessCbHdl

  cleanup callback

  Called while unwinding after a GPI callback
  We keep the process but desensitize it
  NB: need a way to determine if should leave it sensitized…

Class FliTimerCache

class FliTimerCache

  Find the root handle

  Find the root handle using an optional name
  Get a handle to the root simulator object. This is usually the toplevel.
  If no name is provided, we return the first root instance.
  If name is provided, we check the name against the available objects until we find a match. If no match is found we return NULL

Class FliValueObjHdl

class FliValueObjHdl : public FliSignalObjHdl
  Subclassed by FliEnumObjHdl, FliIntObjHdl, FliLogicObjHdl, FliRealObjHdl, FliStringObjHdl
Class GpiCbHdl

class GpiCbHdl : public GpiHdl
    Subclassed by FliProcessCbHdl, GpiValueCbHdl, VhpiCbHdl, VpiCbHdl

Class GpiClockHdl

class GpiClockHdl

Class GpiHdl

class GpiHdl
    Subclassed by GpiCbHdl, GpiIterator, GpiObjHdl

Class GpiImplInterface

class GpiImplInterface
    Subclassed by FliImpl, VhpiImpl, VpiImpl

Class GpiIterator

class GpiIterator : public GpiHdl
    Subclassed by FliIterator, VhpiIterator, VpiIterator, VpiSingleIterator

Class GpiIteratorMapping

template<class Ti, class Tm>
class GpiIteratorMapping

Class GpiObjHdl

class GpiObjHdl : public GpiHdl
    Subclassed by FliObjHdl, GpiSignalObjHdl, VhpiArrayObjHdl, VhpiObjHdl, VpiArrayObjHdl, VpiObjHdl

Class GpiSignalObjHdl

class GpiSignalObjHdl : public GpiObjHdl
    Subclassed by FliSignalObjHdl, VhpiSignalObjHdl, VpiSignalObjHdl
Class GpiValueCbHdl

```cpp
class GpiValueCbHdl : public virtual GpiCbHdl
```

Subclassed by FliSignalCbHdl, VhpiValueCbHdl, VpiValueCbHdl

Class VhpiArrayObjHdl

```cpp
class VhpiArrayObjHdl : public GpiObjHdl
```

Class VhpiCbHdl

```cpp
class VhpiCbHdl : public virtual GpiCbHdl
```

Subclassed by VhpiNextPhaseCbHdl, VhpiReadOnlyCbHdl, VhpiReadWriteCbHdl, VhpiShutdownCbHdl, VhpiStartupCbHdl, VhpiTimedCbHdl, VhpiValueCbHdl

Class VhpiImpl

```cpp
class VhpiImpl : public GpiImplInterface
```

Class VhpiIterator

```cpp
class VhpiIterator : public GpiIterator
```

Class VhpiLogicSignalObjHdl

```cpp
class VhpiLogicSignalObjHdl : public VhpiSignalObjHdl
```

Class VhpiNextPhaseCbHdl

```cpp
class VhpiNextPhaseCbHdl : public VhpiCbHdl
```

Class VhpiObjHdl

```cpp
class VhpiObjHdl : public GpiObjHdl
```

Class VhpiReadOnlyCbHdl

```cpp
class VhpiReadOnlyCbHdl : public VhpiCbHdl
```

Class VhpiReadWriteCbHdl

```cpp
class VhpiReadWriteCbHdl : public VhpiCbHdl
```

Class VhpiShutdownCbHdl

```cpp
class VhpiShutdownCbHdl : public VhpiCbHdl
```

Class VhpiStartupCbHdl

```cpp
class VhpiStartupCbHdl : public VhpiCbHdl
```

Class VhpiTimedCbHdl

```cpp
class VhpiTimedCbHdl : public VhpiCbHdl
```

Class VhpiValueCbHdl

```cpp
class VhpiValueCbHdl : public VhpiCbHdl
```
Class VhpiReadwriteCbHdl

class VhpiReadwriteCbHdl : public VhpiCbHdl

Class VhpiShutdownCbHdl

class VhpiShutdownCbHdl : public VhpiCbHdl

Class VhpiSignalObjHdl

class VhpiSignalObjHdl : public GpiSignalObjHdl
    Subclassed by VhpiLogicSignalObjHdl

Class VhpiStartupCbHdl

class VhpiStartupCbHdl : public VhpiCbHdl

Class VhpiTimedCbHdl

class VhpiTimedCbHdl : public VhpiCbHdl

Class VhpiValueCbHdl

class VhpiValueCbHdl : public VhpiCbHdl, public GpiValueCbHdl

Class VpiArrayObjHdl

class VpiArrayObjHdl : public GpiObjHdl

Class VpiCbHdl

class VpiCbHdl : public virtual GpiCbHdl
    Subclassed by VpiNextPhaseCbHdl, VpiReadOnlyCbHdl, VpiReadwriteCbHdl, VpiShutdownCbHdl, VpiStartupCbHdl, VpiTimedCbHdl, VpiValueCbHdl

Class VpiImpl

class VpiImpl : public GpiImplInterface
Class VpiIterator

```cpp
class VpiIterator : public GpiIterator
```

Class VpiNextPhaseCbHdl

```cpp
class VpiNextPhaseCbHdl : public VpiCbHdl
```

Class VpiObjHdl

```cpp
class VpiObjHdl : public GpiObjHdl
```

Class VpiReadOnlyCbHdl

```cpp
class VpiReadOnlyCbHdl : public VpiCbHdl
```

Class VpiReadWriteCbHdl

```cpp
class VpiReadWriteCbHdl : public VpiCbHdl
```

Class VpiShutdownCbHdl

```cpp
class VpiShutdownCbHdl : public VpiCbHdl
```

Class VpiSignalObjHdl

```cpp
class VpiSignalObjHdl : public GpiSignalObjHdl
```

Class VpiSingleIterator

```cpp
class VpiSingleIterator : public GpiIterator
```

Class VpiStartupCbHdl

```cpp
class VpiStartupCbHdl : public VpiCbHdl
```

Class VpiTimedCbHdl

```cpp
class VpiTimedCbHdl : public VpiCbHdl
```
Class VpiValueCbHdl

class VpiValueCbHdl : public VpiCbHdl, public GpiValueCbHdl

Class cocotb_entrypoint

class cocotb_entrypoint

Class cocotb_entrypoint::cocotb_arch

class cocotb_arch

8.1.2 File list

File FliCbHdl.cpp

File FliImpl.cpp

Find the root handle

Find the root handle using an optional name
Get a handle to the root simulator object. This is usually the toplevel.
If no name is provided, we return the first root instance.
If name is provided, we check the name against the available objects until we find a match. If no match is found we return NULL

void fli_mappings (GpiliteratorMapping<int, FliIterator::OneToMany> &map)

void handle_fli_callback (void *data)

static void register_initial_callback ()

static void register_final_callback ()

static void register_embed ()

void cocotb_init ()

GPI_ENTRY_POINT (fli, register_embed)

Variables

FliProcessCbHdl *sim_init_cb

FliProcessCbHdl *sim_finish_cb

FliImpl *fli_table
File FliImpl.h

Functions

void cocotb_init()

void handle_fli_callback(void *data)

class FliProcessCbHdl: public virtual GpiCbHdl
    Subclassed by FliShutdownCbHdl, FliSignalCbHdl, FliSimPhaseCbHdl, FliStartupCbHdl, FliTimedCbHdl

    cleanup callback
    Called while unwinding after a GPI callback
    We keep the process but desensitize it
    NB: need a way to determine if should leave it sensitized...

    int cleanup_callback()

Public Functions

    FliProcessCbHdl(GpiImplInterface *impl)
    virtual ~FliProcessCbHdl()
    virtual int arm_callback() = 0

Protected Attributes

    mtiProcessIdT m_proc_hdl
    bool m_sensitised

class FliSignalCbHdl: public FliProcessCbHdl, public GpiValueCbHdl

    cleanup callback
    Called while unwinding after a GPI callback
    We keep the process but desensitize it
    NB: need a way to determine if should leave it sensitized...

    FliSignalCbHdl(GpiImplInterface *impl, FliSignalObjHdl *sig_hdl, unsigned int edge)
    int arm_callback()
Public Functions

virtual ~FliSignalCbHdl()
int cleanup_callback()

Private Members

mtiSignalIdT m_sig_hdl

class FliSimPhaseCbHdl : public FliProcessCbHdl
Subclassed by FliNextPhaseCbHdl, FliReadOnlyCbHdl, FliReadWriteCbHdl

cleanup callback

Called while unwinding after a GPI callback
We keep the process but desensitize it
NB: need a way to determine if should leave it sensitized...
int arm_callback()

Public Functions

FliSimPhaseCbHdl (GpiImplInterface *impl, mtiProcessPriorityT priority)
virtual ~FliSimPhaseCbHdl()

Protected Attributes

mtiProcessPriorityT m_priority

class FliReadWriteCbHdl : public FliSimPhaseCbHdl

Public Functions

FliReadWriteCbHdl (GpiImplInterface *impl)
virtual ~FliReadWriteCbHdl()

class FliNextPhaseCbHdl : public FliSimPhaseCbHdl

Public Functions

FliNextPhaseCbHdl (GpiImplInterface *impl)
virtual ~FliNextPhaseCbHdl()

class FliReadOnlyCbHdl : public FliSimPhaseCbHdl
Public Functions

FliReadOnlyCbHdl (GpiImplInterface *impl)
virtual ~FliReadOnlyCbHdl ()
class FliStartupCbHdl : public FliProcessCbHdl

cleanup callback
Called while unwinding after a GPI callback
We keep the process but desensitize it
NB: need a way to determine if should leave it sensitized...
int arm_callback ()
int run_callback ()

Public Functions

FliStartupCbHdl (GpiImplInterface *impl)
virtual ~FliStartupCbHdl ()
class FliShutdownCbHdl : public FliProcessCbHdl

cleanup callback
Called while unwinding after a GPI callback
We keep the process but desensitize it
NB: need a way to determine if should leave it sensitized...
int arm_callback ()
int run_callback ()

Public Functions

FliShutdownCbHdl (GpiImplInterface *impl)
virtual ~FliShutdownCbHdl ()
class FliTimedCbHdl : public FliProcessCbHdl
cleanup callback

Called while unwinding after a GPI callback
We keep the process but desensitize it
NB: need a way to determine if should leave it sensitized...

FliTimedCbHdl (GpiImplInterface *impl, uint64_t time_ps)
int arm_callback ()
int cleanup_callback ()

Public Functions

virtual ~FliTimedCbHdl ()
void reset_time (uint64_t new_time)

Private Members

uint64_t m_time_ps

class FliObj
Subclassed by FliObjHdl, FliSignalObjHdl

Public Functions

FliObj (int acc_type, int acc_full_type)
virtual ~FliObj ()
int get_acc_type ()
int get_acc_full_type ()

Protected Attributes

int m_acc_type
int m_acc_full_type

class FliObjHdl: public GpiObjHdl, public FliObj

Public Functions

FliObjHdl (GpiImplInterface *impl, void *hdl, gpi_objtype_t objtype, int acc_type, int acc_full_type)
FliObjHdl (GpiImplInterface *impl, void *hdl, gpi_objtype_t objtype, int acc_type, int acc_full_type, bool is_const)
virtual ~FliObjHdl ()
int initialise (std::string &name, std::string &fq_name)
class FliSignalObjHdl : public GpiSignalObjHdl, public FliObj
Subclassed by FliValueObjHdl

Public Functions

FliSignalObjHdl (GpiImplInterface *impl, void *hdl, gpi_objtype_t objtype, bool is_const, int acc_type, int acc_full_type, bool is_var)

virtual ~FliSignalObjHdl ()

GpiCbHdl *value_change_cb (unsigned int edge)

int initialise (std::string &name, std::string &fq_name)

bool is_var ()

Protected Attributes

bool m_is_var
FliSignalCbHdl m_rising_cb
FliSignalCbHdl m_falling_cb
FliSignalCbHdl m_either_cb

class FliValueObjHdl : public FliSignalObjHdl
Subclassed by FliEnumObjHdl, FliIntObjHdl, FliLogicObjHdl, FliRealObjHdl, FliStringObjHdl

Public Functions

FliValueObjHdl (GpiImplInterface *impl, void *hdl, gpi_objtype_t objtype, bool is_const, int acc_type, int acc_full_type, bool is_var, mtiTypeIdT valType, mtiTypeKindT type-Kind)

virtual ~FliValueObjHdl ()

const char *get_signal_value_binstr ()

const char *get_signal_value_str ()

double get_signal_value_real ()

long get_signal_value_long ()

int set_signal_value (const long value)

int set_signal_value (const double value)

int set_signal_value (std::string &value)

void *get_sub_hdl (int index)

int initialise (std::string &name, std::string &fq_name)

mtiTypeKindT get_fli_typekind ()

mtiTypeIdT get_fli_typeid ()

8.1. API Documentation
Protected Attributes

mtiTypeKindT m_fli_type
mtiTypeIdT m_val_type
char *m_val_buff
void **m_sub_hdls

class FliEnumObjHdl: public FliValueObjHdl

Public Functions

FliEnumObjHdl(GpiImplInterface *impl, void *hdl, gpi_objtype_t objtype, bool is_const, int acc_type, int acc_full_type, bool is_var, mtiTypeIdT valType, mtiTypeKindT type-Kind)

virtual ~FliEnumObjHdl() const

const char *get_signal_value_str()

long get_signal_value_long()

int set_signal_value(const long value)

int initialise(std::string &name, std::string &fq_name)

Private Members

char **m_value_enum
mtiInt32T m_num_enum

class FliLogicObjHdl: public FliValueObjHdl

Public Functions

FliLogicObjHdl(GpiImplInterface *impl, void *hdl, gpi_objtype_t objtype, bool is_const, int acc_type, int acc_full_type, bool is_var, mtiTypeIdT valType, mtiTypeKindT type-Kind)

virtual ~FliLogicObjHdl() const

const char *get_signal_value_binstr()

int set_signal_value(const long value)

int set_signal_value(std::string &value)

int initialise(std::string &name, std::string &fq_name)
Private Members

char *m_mti_buff
char **m_value_enum
mtiInt32T m_num_enum
std::map<char, mtiInt32T> m_enum_map

class FliIntObjHdl : public FliValueObjHdl

Public Functions

FliIntObjHdl (GpiImplInterface *impl, void *hdl, gpi_objtype_t objtype, bool is_const, int acc_type,
int acc_full_type, bool is_var, mtiTypeIdT valType, mtiTypeKindT typeKind)
virtual ~FliIntObjHdl ()
const char *get_signal_value_binstr ()
long get_signal_value_long ()
itn set_signal_value (const long value)
itn initialise (std::string &name, std::string &fq_name)

class FliRealObjHdl : public FliValueObjHdl

Public Functions

FliRealObjHdl (GpiImplInterface *impl, void *hdl, gpi_objtype_t objtype, bool is_const, int acc_type,
int acc_full_type, bool is_var, mtiTypeIdT valType, mtiTypeKindT typeKind)
virtual ~FliRealObjHdl ()
double get_signal_value_real ()
itn set_signal_value (const double value)
itn initialise (std::string &name, std::string &fq_name)

Private Members

double *m_mti_buff

class FliStringObjHdl : public FliValueObjHdl
Public Functions

FliStringObjHdl (GpiImplInterface *impl, void *hdl, gpi_objtype_t objtype, bool is_const, int acc_type, int acc_full_type, bool is_var, mtiTypeIdT valType, mtiTypeKindT typeKind)

t~FliStringObjHdl()

const char *get_signal_value_str()

int set_signal_value (std::string &value)

int initialise (std::string &name, std::string &fq_name)

Private Members

char *m_mti_buff

class FliTimerCache

Find the root handle

Find the root handle using an optional name

Get a handle to the root simulator object. This is usually the toplevel.

If no name is provided, we return the first root instance.

If name is provided, we check the name against the available objects until we find a match. If no match is found we return NULL.

FliTimedCbHdl *get_timer (uint64_t time_ps)

void put_timer (FliTimedCbHdl *hdl)

Public Functions

FliTimerCache (Fliimpl *impl)

t~FliTimerCache()

Private Members

std::queue<FliTimedCbHdl *> free_list

Fliimpl *impl

class FliIterator : public GpiIterator
Find the root handle

Find the root handle using an optional name
Get a handle to the root simulator object. This is usually the toplevel.
If no name is provided, we return the first root instance.
If name is provided, we check the name against the available objects until we find a match. If no match is found
we return NULL.

\textit{GpiIteratorMapping< int, FliIterator::OneToMany> iterate_over}

\begin{verbatim}
FliIterator (GpiImplInterface *impl, GpiObjHdl *hdl)

GpiIterator::Status next_handle (std::string &name, GpiObjHdl **hdl, void **raw_hdl)

void populate_handler_list (OneToMany childType)
\end{verbatim}

Public Types

\begin{verbatim}
enumOneToMany
    Values:
        OTM_END = 0
        OTM_CONSTANTS
        OTM_SIGNALS
        OTM_REGIONS
        OTM_SIGNAL_SUB_ELEMENTS
        OTM_VARIABLE_SUB_ELEMENTS
\end{verbatim}

Public Functions

\begin{verbatim}
virtual ~FliIterator()
\end{verbatim}

Private Members

\begin{verbatim}
std::vector<OneToMany> *selected
std::vector<OneToMany>::iterator one2many
std::vector<void *> m_vars
std::vector<void *> m_sigs
std::vector<void *> m_regs
std::vector<void *> *m_currentHandles
std::vector<void *> ::iterator m_iterator
\end{verbatim}

\textbf{class FliImpl : public GpiImplInterface}
Native Check Create

Determine whether a simulation object is native to FLI and create a handle if it is

\[ \text{GpiObjHdl} \ast \text{native_check_create} \left( \text{std::string} \ & \text{name}, \text{GpiObjHdl} \ast \text{parent} \right) \]

\[ \text{GpiObjHdl} \ast \text{native_check_create} \left( \text{int32}_t \ \text{index}, \text{GpiObjHdl} \ast \text{parent} \right) \]

\[ \text{const char} \ast \text{reason_to_string} \left( \text{int reason} \right) \]

Get current simulation time

Get current simulation time
NB units depend on the simulation configuration

\[ \text{void get_sim_time} \left( \text{uint32}_t \ast \text{high}, \text{uint32}_t \ast \text{low} \right) \]

\[ \text{void get_sim_precision} \left( \text{int32}_t \ast \text{precision} \right) \]

Find the root handle

Find the root handle using an optional name

Get a handle to the root simulator object. This is usually the toplevel.
If no name is provided, we return the first root instance.
If name is provided, we check the name against the available objects until we find a match. If no match is found we return NULL

\[ \text{GpiObjHdl} \ast \text{get_root_handle} \left( \text{const char} \ast \text{name} \right) \]

\[ \text{Gpilte}rator \ast \text{iterate_handle} \left( \text{GpiObjHdl} \ast \text{obj_hdl}, \text{gpi_iterator_sel}_t \text{type} \right) \]

\[ \text{GpiCbHdl} \ast \text{register_timed_callback} \left( \text{uint64}_t \ast \text{time_ps} \right) \]

\[ \text{GpiCbHdl} \ast \text{register_readonly_callback} \left( \right) \]

\[ \text{GpiCbHdl} \ast \text{register_nexttime_callback} \left( \right) \]

\[ \text{GpiCbHdl} \ast \text{register_readwrite_callback} \left( \right) \]

\[ \text{int deregister_callback} \left( \text{GpiCbHdl} \ast \text{obj_hdl} \right) \]

Public Functions

\[ \text{FliImpl} \left( \text{const std::string} \ & \text{name} \right) \]

\[ \text{void sim_end} \left( \right) \]

\[ \text{GpiObjHdl} \ast \text{native_check_create} \left( \text{void} \ast \text{raw_hdl}, \text{GpiObjHdl} \ast \text{paret} \right) \]

\[ \text{GpiObjHdl} \ast \text{create_gpi_obj_from_handle} \left( \text{void} \ast \text{hdl}, \text{std::string} \ & \text{name}, \text{std::string} \ & \text{fq_name}, \right. \]
\[ \left. \text{int accType}, \text{int accFullType} \right) \]
Public Members

_FliTimerCache cache_

Private Functions

bool _isValueConst_ (int kind)
bool _isValueLogic_ (mtiTypeIdT type)
bool _isValueChar_ (mtiTypeIdT type)
bool _isValueBoolean_ (mtiTypeIdT type)
bool _isTypeValue_ (int type)
bool _isTypeSignal_ (int type, int full_type)

Private Members

_FliReadOnlyCbHdl m_readonly_cbhdl_
_FliNextPhaseCbHdl m_nexttime_cbhdl_
_FliReadWriteCbHdl m_readwrite_cbhdl_

File FliObjHdl.cpp

File GpiCbHdl.cpp

Defines

    ret = X; break ]

File GpiCommon.cpp

Defines

    CHECK_AND_STORE (x) x
    CLEAR_STORE () (void)0
    DOT_LIB_EXT ".” xstr(LIB_EXT)

Functions

int gpi_print_registered_impl ()
int gpi_register_impl (GpiImplInterface *func_tbl)
void gpi_embed_init (gpi_sim_info_t *info)
void gpi_embed_end ()
void gpi_sim_end ()
void gpi_cleanup (void)
void gpi_embed_event (gpi_event_t level, const char *msg)
static void gpi_load_libs (std::vector<std::string> to_load)
void gpi_load_extra_libs ()
void gpi_get_sim_time (uint32_t *high, uint32_t *low)
void gpi_get_sim_precision (int32_t *precision)
gpi_sim_hdl gpi_get_root_handle (const char *name)
static GpiObjHdl *__gpi_get_handle_by_name (GpiObjHdl *parent, std::string name, GpiImplInterface *skip_impl)
static GpiObjHdl *__gpi_get_handle_by_raw (GpiObjHdl *parent, void *raw_hdl, GpiImplInterface *skip_impl)
gpi_sim_hdl gpi_get_handle_by_name (gpi_sim_hdl parent, const char *name)
gpi_sim_hdl gpi_get_handle_by_index (gpi_sim_hdl parent, int32_t index)
gpi_iterator_hdl gpi_iterate (gpi_sim_hdl base, gpi_iterator_sel_t type)
gpi_sim_hdl gpi_next (gpi_iterator_hdl iterator)
const char *gpi_get_definition_name (gpi_sim_hdl sig_hdl)
const char *gpi_get_definition_file (gpi_sim_hdl sig_hdl)
const char *gpi_get_signal_value_binstr (gpi_sim_hdl sig_hdl)
const char *gpi_get_signal_value_str (gpi_sim_hdl sig_hdl)
double gpi_get_signal_value_real (gpi_sim_hdl sig_hdl)
long gpi_get_signal_value_long (gpi_sim_hdl sig_hdl)
const char *gpi_get_signal_name_str (gpi_sim_hdl sig_hdl)
const char *gpi_get_signal_type_str (gpi_sim_hdl sig_hdl)
gpi_objtype_t gpi_get_object_type (gpi_sim_hdl sig_hdl)
int gpi_is_constant (gpi_sim_hdl sig_hdl)
int gpi_is_indexable (gpi_sim_hdl sig_hdl)
void gpi_set_signal_value_long (gpi_sim_hdl sig_hdl, long value)
void gpi_set_signal_value_str (gpi_sim_hdl sig_hdl, const char *str)
void gpi_set_signal_value_real (gpi_sim_hdl sig_hdl, double value)
int gpi_get_num_elems (gpi_sim_hdl sig_hdl)
int gpi_get_range_left (gpi_sim_hdl sig_hdl)
int gpi_get_range_right (gpi_sim_hdl sig_hdl)
gpi_sim_hdl gpi_register_value_change_callback (int (*gpi_function)) const void *
    , void *gpi_cb_data, gpi_sim_hdl sig_hdl, unsigned int edge

gpi_sim_hdl gpi_register_timed_callback (int (*gpi_function)) const void *
    , void *gpi_cb_data, uint64_t time_ps

gpi_sim_hdl gpi_register_readonly_callback (int (*gpi_function)) const void *
    , void *gpi_cb_data
```c
void gpi_stop_clock(gpi_sim_hdl clk_object)

void gpi_deregister_callback(gpi_sim_hdl hdl)

Variables

vector<GpiImplInterface *> registered_impls

File VhpiCbHdl.cpp

Defines

VHPI_TYPE_MIN (1000)

Functions

void handle_vhpi_callback(const vhpiCbDataT *cb_data)

bool get_range(vhpiHandleT hdl, vhpiIntT dim, int *left, int *right)

void vhpi_mappings(GpiIteratorMapping<vhpiClassKindT, vhpiOneToManyT> &map)

File Vhpilmpl.cpp

Defines

CASE_STR (_X) case _X: return #_X

Functions

bool is_const(vhpiHandleT hdl)

bool is_enum_logic(vhpiHandleT hdl)

bool is_enum_char(vhpiHandleT hdl)

bool is_enum_boolean(vhpiHandleT hdl)

void handle_vhpi_callback(const vhpiCbDataT *cb_data)

static void register_initial_callback()

static void register_final_callback()

static void register_embed()

void vhpi_startup_routines_bootstrap()
```
Variables

VhpiCbHdl *sim_init_cb
VhpiCbHdl *sim_finish_cb
VhpiImpl *vhpi_table

void (*vhpi_start-up_routines[])() = { register_embed, . . . }

File VhpiImpl.h

Defines

GEN_IDX_SEP_LHS "__"
GEN_IDX_SEP_RHS ""

__check_vhpi_error(__FILE__, __func__, __LINE__); } while (0) ]

Functions

static int __check_vhpi_error (const char *file, const char *func, long line)

class VhpiCbHdl : public virtual GpiCbHdl

Subclassed by VhpiNextPhaseCbHdl, VhpiReadOnlyCbHdl, VhpiReadWriteCbHdl, VhpiShutdownCbHdl, VhpiStartupCbHdl, VhpiTimedCbHdl, VhpiValueCbHdl

Public Functions

VhpiCbHdl (GpiImplInterface *impl)

virtual ~VhpiCbHdl ()

int arm_callback ()

int cleanup_callback ()

Protected Attributes

vhpiciDataT cb_data

vhpiTimeT vhpi_time

class VhpiValueCbHdl : public VhpiCbHdl, public GpiValueCbHdl
Public Functions

VhpiValueCbHdl(GpiImplInterface *impl, VhpiSignalObjHdl *sig, int edge)

virtual ~VhpiValueCbHdl()

int cleanup_callback()

Private Members

std::string initial_value
bool rising
bool falling
VhpiSignalObjHdl *signal

class VhpiTimedCbHdl: public VhpiCbHdl

Public Functions

VhpiTimedCbHdl(GpiImplInterface *impl, uint64_t time_ps)

virtual ~VhpiTimedCbHdl()

int cleanup_callback()

class VhpiReadOnlyCbHdl: public VhpiCbHdl

Public Functions

VhpiReadOnlyCbHdl(GpiImplInterface *impl)

virtual ~VhpiReadOnlyCbHdl()

class VhpiNextPhaseCbHdl: public VhpiCbHdl

Public Functions

VhpiNextPhaseCbHdl(GpiImplInterface *impl)

virtual ~VhpiNextPhaseCbHdl()

class VhpiStartupCbHdl: public VhpiCbHdl
Public Functions

VhpiStartupCbHdl (GpiImplInterface *impl)
int run_callback()
int cleanup_callback()
virtual ~VhpiStartupCbHdl()

class VhpiStartupCbHdl : public VhpiCbHdl

Public Functions

VhpiShutdownCbHdl (GpiImplInterface *impl)
int run_callback()
int cleanup_callback()
virtual ~VhpiShutdownCbHdl()

class VhpiShutdownCbHdl : public VhpiCbHdl

Public Functions

VhpiReadwriteCbHdl (GpiImplInterface *impl)
virtual ~VhpiReadwriteCbHdl()

class VhpiReadwriteCbHdl : public VhpiCbHdl

Public Functions

VhpiArrayObjHdl (GpiImplInterface *impl, vhpiHandleT hdl, gpi_objtype_t objtype)
~VhpiArrayObjHdl()
int initialise (std::string &name, std::string &fq_name)

class VhpiArrayObjHdl : public GpiObjHdl

Public Functions

VhpiObjHdl (GpiImplInterface *impl, vhpiHandleT hdl, gpi_objtype_t objtype)
~VhpiObjHdl()
int initialise (std::string &name, std::string &fq_name)

class VhpiObjHdl : public GpiObjHdl

Public Functions

VhpiSignalObjHdl (GpiImplInterface *impl, vhpiHandleT hdl, gpi_objtype_t objtype)
~VhpiSignalObjHdl()
int initialise (std::string &name, std::string &fq_name)

class VhpiSignalObjHdl : public GpiSignalObjHdl
Subclassed by VhpiLogicSignalObjHdl
Public Functions

VhpiSignalObjHdl (GpiImplInterface *impl, vhpiHandleT hdl, gpi_objtype_t objtype, bool is_const)
~VhpiSignalObjHdl()
const char *get_signal_value_binstr()
const char *get_signal_value_str()
double get_signal_value_real()
long get_signal_value_long()
int set_signal_value (const long value)
int set_signal_value (const double value)
int set_signal_value (std::string &value)
GpiCbHdl *value_change_cb (unsigned int edge)
int initialise (std::string &name, std::string &fq_name)

Protected Functions

const vhpiEnumT chr2vhpi (const char value)

Protected Attributes

vhpiValueT m_value
vhpiValueT m_binvalue
VhpiValueCbHdl m_rising_cb
VhpiValueCbHdl m_falling_cb
VhpiValueCbHdl m_either_cb

class VhpiLogicSignalObjHdl : public VhpiSignalObjHdl

Public Functions

VhpiLogicSignalObjHdl (GpiImplInterface *impl, vhpiHandleT hdl, gpi_objtype_t objtype, bool is_const)
virtual ~VhpiLogicSignalObjHdl()
int set_signal_value (const long value)
int set_signal_value (std::string &value)
int initialise (std::string &name, std::string &fq_name)

class VhpiIterator : public Gpiliterator

8.1. API Documentation
Public Functions

VhpiIterator (GpiImplInterface *impl, GpiObjHdl *hdl)

~VhpiIterator()

GpilIterator::Status next_handle (std::string &name, GpiObjHdl **hdl, void **raw_hdl)

Private Members

vhpiHandleT m_iterator
vhpiHandleT m_iter_obj
std::vector<vhpiOneToManyT> *selected
std::vector<vhpiOneToManyT>::iterator one2many

Private Static Attributes

GpilIteratorMapping<vhpiClassKindT, vhpiOneToManyT> iterate_over

class VhpiImpl: public GpiImplInterface

Public Functions

VhpiImpl (const std::string &name)

void sim_end()

void get_sim_time (uint32_t *high, uint32_t *low)

void get_sim_precision (int32_t *precision)

GpiObjHdl *get_root_handle (const char *name)

GpilIterator *iterate_handle (GpiObjHdl *obj_hdl, gpi_iterator_sel_t type)

GpiCbHdl *register_timed_callback (uint64_t time_ps)

GpiCbHdl *register_readonly_callback ()

GpiCbHdl *register_nexttime_callback ()

GpiCbHdl *register_readwrite_callback ()

int deregister_callback (GpiCbHdl *obj_hdl)

GpiObjHdl *native_check_create (std::string &name, GpiObjHdl *parent)

GpiObjHdl *native_check_create (int32_t index, GpiObjHdl *parent)

GpiObjHdl *native_check_create (void *raw_hdl, GpiObjHdl *parent)

const char *reason_to_string (int reason)

const char *format_to_string (int format)
**GpiObjHdl** *create_gpi_obj_from_handle*(vhpiHandleT new_hdl, std::string &name, std::string &fq_name)

**Private Members**

VhpiReadWriteCbHdl m_read_write
VhpiNextPhaseCbHdl m_next_phase
VhpiReadOnlyCbHdl m_read_only

File VpiCbHdl.cpp

**Defines**

VPI_TYPE_MAX (1000)

**Functions**

int32_t handle_vpi_callback (p_cb_data cb_data)

void vpi_mappings (GpilIteratorMapping<int32_t, int32_t> &map)

File Vpimpl.cpp

**Defines**

CASE_STR (_X) case _X: return #_X

**Functions**

gpi_objtype_t to_gpi_objtype (int32_t vpitype)

int32_t handle_vpi_callback (p_cb_data cb_data)

static void register_embed()

static void register_initial_callback()

static void register_final_callback()

static int system_function_compilef (char *userdata)

static int system_function_overload (char *userdata)

static void register_system_functions()

void vlog_startup_routines_bootstrap()
Variables

\[ VpiCbHdl \texttt{*sim\_init\_cb} \]
\[ VpiCbHdl \texttt{*sim\_finish\_cb} \]
\[ VpiImpl \texttt{*vpi\_table} \]

\[
\begin{align*}
\text{int systf\_info\_level} & = \texttt{GPIInfo} \\
\text{int systf\_warning\_level} & = \texttt{GPIWarning} \\
\text{int systf\_error\_level} & = \texttt{GPIError} \\
\text{int systf\_fatal\_level} & = \texttt{GPICritical} \\
\end{align*}
\]

\[
\text{void (*vlog\_startup\_routines[])(}) = \{ \texttt{register\_embed}, \ldots \}\]

File \texttt{VpiImpl.h}

Defines

\[
\text{\_check\_vpi\_error(__FILE__, \_func__, \_LINE__); } \text{while (0)} \]

Functions

static int \texttt{\_check\_vpi\_error(const char *file, const char *func, long line)}

class \texttt{VpiCbHdl : public virtual GpiCbHdl}

\text{Subclassed by VpiNextPhaseCbHdl, VpiReadOnlyCbHdl, VpiReadWriteCbHdl, VpiShutdownCbHdl, VpiStartupCbHdl, VpiTimedCbHdl, VpiValueCbHdl}

Public Functions

\texttt{VpiCbHdl(GpiImplInterface \texttt{*impl})}

virtual \texttt{~VpiCbHdl()}

int \texttt{arm\_callback()}

int \texttt{cleanup\_callback()}

Protected Attributes

\texttt{s\_cb\_data \ \textbf{cb\_data}}

\texttt{s\_vpi\_time \ \textbf{vpi\_time}}

class \texttt{VpiValueCbHdl : public VpiCbHdl, public GpiValueCbHdl}
Public Functions

VpiValueCbHdl (GpiImplInterface *impl, VpiSignalObjHdl *sig, int edge)
virtual ~VpiValueCbHdl ()
int cleanup_callback ()

Private Members

s_vpi_value m_vpi_value

class VpiTimedCbHdl: public VpiCbHdl

Public Functions

VpiTimedCbHdl (GpiImplInterface *impl, uint64_t time_ps)
virtual ~VpiTimedCbHdl ()
int cleanup_callback ()

class VpiReadOnlyCbHdl: public VpiCbHdl

Public Functions

VpiReadOnlyCbHdl (GpiImplInterface *impl)
virtual ~VpiReadOnlyCbHdl ()

class VpiNextPhaseCbHdl: public VpiCbHdl

Public Functions

VpiNextPhaseCbHdl (GpiImplInterface *impl)
virtual ~VpiNextPhaseCbHdl ()

class VpiReadWriteCbHdl: public VpiCbHdl

Public Functions

VpiReadWriteCbHdl (GpiImplInterface *impl)
virtual ~VpiReadWriteCbHdl ()

class VpiStartupCbHdl: public VpiCbHdl
Public Functions

VpiStartupCbHdl (GpiImplInterface *impl)
int run_callback()
int cleanup_callback()
virtual ~VpiStartupCbHdl()
class VpiShutdownCbHdl : public VpiCbHdl

Public Functions

VpiShutdownCbHdl (GpiImplInterface *impl)
int run_callback()
int cleanup_callback()
virtual ~VpiShutdownCbHdl()
class VpiArrayObjHdl : public GpiObjHdl

Public Functions

VpiArrayObjHdl (GpiImplInterface *impl, vpiHandle hdl, gpi_objtype_t objtype)
virtual ~VpiArrayObjHdl()
int initialise (std::string &name, std::string &fq_name)
class VpiObjHdl : public GpiObjHdl

Public Functions

VpiObjHdl (GpiImplInterface *impl, vpiHandle hdl, gpi_objtype_t objtype)
virtual ~VpiObjHdl()
int initialise (std::string &name, std::string &fq_name)
class VpiSignalObjHdl : public GpiSignalObjHdl

Public Functions

VpiSignalObjHdl (GpiImplInterface *impl, vpiHandle hdl, gpi_objtype_t objtype, bool is_const)
virtual ~VpiSignalObjHdl()
const char *get_signal_value_binstr()
const char *get_signal_value_str()
double get_signal_value_real()
long get_signal_value_long()

int set_signal_value(const long value)
int set_signal_value(const double value)
int set_signal_value(const std::string &value)

GpiCbHdl *value_change_cb (unsigned int edge)

int initialise (std::string &name, std::string &fq_name)

Private Functions

int set_signal_value(s_vpi_value value)

Private Members

VpiValueCbHdl m_rising_cb
VpiValueCbHdl m_falling_cb
VpiValueCbHdl m_either_cb

class VpiIterator: public GpiIterator

Public Functions

VpiIterator (GpiImplInterface *impl, GpiObjHdl *hdl)
~VpiIterator ()

Gpiliterator::Status next_handle (std::string &name, GpiObjHdl **hdl, void **raw_hdl)

Private Members

vpiHandle m_iterator
std::vector<int32_t> *selected
std::vector<int32_t>::iterator one2many

Private Static Attributes

GpiIteratorMapping<int32_t, int32_t> iterate_over

class VpiSingleIterator: public GpiIterator
Public Functions

VpiSingletonIterator (GpiImplInterface *impl, GpiObjHdl *hdl, int32_t vpitype)

virtual ~VpiSingletonIterator ()

GpIterator::Status next_handle (std::string &name, GpiObjHdl **hdl, void **raw_hdl)

Protected Attributes

vpiHandle m_iterator

class VpiImpl : public GpiImplInterface

Public Functions

VpiImpl (const std::string &name)

void sim_end ()

void get_sim_time (uint32_t *high, uint32_t *low)

void get_sim_precision (int32_t *precision)

GpiObjHdl *get_root_handle (const char *name)

GpIterator *iterate_handle (GpiObjHdl *obj_hdl, gpi_iterator_sel_t type)

GpiObjHdl *next_handle (GpIterator *iter)

GpiCbHdl *register_timed_callback (uint64_t time_ps)

GpiCbHdl *register_readonly_callback ()

GpiCbHdl *register_nexttime_callback ()

GpiCbHdl *register_readwrite_callback ()

int deregister_callback (GpiCbHdl *obj_hdl)

GpiObjHdl *native_check_create (std::string &name, GpiObjHdl *parent)

GpiObjHdl *native_check_create (int32_t index, GpiObjHdl *parent)

GpiObjHdl *native_check_create (void *raw_hdl, GpiObjHdl *parent)

const char *reason_to_string (int reason)

GpiObjHdl *create_gpi_obj_from_handle (vpiHandle new_hdl, std::string &name, std::string &fq_name)
Private Members

VpiReadWriteCbHdl m_read_write
VpiNextPhaseCbHdl m_next_phase
VpiReadOnlyCbHdl m_read_only

File cocotb_utils.c

Functions

void to_python (void)
void to_simulator (void)
void *utils_dyn_open (const char *lib_name)
void *utils_dyn_sym (void *handle, const char *sym_name)

Variables

int is_python_context = 0

File cocotb_utils.h

Defines

Xstr (a) str(a)
str (a) #a

Functions

void *utils_dyn_open (const char *lib_name)
void *utils_dyn_sym (void *handle, const char *sym_name)
void to_python (void)
void to_simulator (void)

Variables

int is_python_context
File embed.h

Functions

void embed_init_python (void)
void embed_sim_cleanup (void)
int embed_sim_init (gpi_sim_info_t *info)
void embed_sim_event (gpi_event_t level, const char *msg)

File entrypoint.vhdl

class cocotb_entrypoint

    class cocotb_arch

        Public Members

            cocotb_arch:architecture is "cocotb_fli_init fli.so" cocotb_entrypoint.cocotb_arch

File gpi.h

 Defines

DLLEXPORT
EXTERN_C_START
EXTERN_C_END
__attribute__ (x)
    return 0; \ else \ return -1 ]

Typedefs

typedef EXTERN_C_START enum gpi_event_e gpi_event_t
typedef struct gpi_sim_info_s gpi_sim_info_t
typedef void *gpi_sim_hdl
typedef void *gpi_iterator_hdl
typedef enum gpi_objtype_e gpi_objtype_t
typedef enum gpi_iterator_sel_e gpi_iterator_sel_t
typedef enum gpi_edge gpi_edge_e
Enums

enum gpi_event_e
Values:
    SIM_INFO = 0
    SIM_TEST_FAIL = 1
    SIM_FAIL = 2

denum gpi_objtype_e
Values:
    GPI_UNKNOWN = 0
    GPI_MEMORY = 1
    GPI_MODULE = 2
    GPI_NET = 3
    GPI_PARAMETER = 4
    GPI_REGISTER = 5
    GPI_ARRAY = 6
    GPI_ENUM = 7
    GPI_STRUCTURE = 8
    GPI_REAL = 9
    GPI_INTEGER = 10
    GPI_STRING = 11
    GPI_GENARRAY = 12

denum gpi_iterator_sel_e
Values:
    GPI_OBJECTS = 1
    GPI_DRIVERS = 2
    GPI_LOADS = 3

denum gpi_edge
Values:
    GPI_RISING = 1
    GPI_FALLING = 2
Functions

void gpi_sim_end (void)
void gpi_cleanup (void)
void gpi_get_sim_time (uint32_t *high, uint32_t *low)
void gpi_get_sim_precision (int32_t *precision)
gpi_sim_hdl gpi_get_root_handle (const char *name)
gpi_sim_hdl gpi_get_handle_by_name (gpi_sim_hdl parent, const char *name)
gpi_sim_hdl gpi_get_handle_by_index (gpi_sim_hdl parent, int32_t index)
void gpi_free_handle (gpi_sim_hdl gpi_hdl)
gpi_iterator_hdl gpi_iterate (gpi_sim_hdl base, gpi_iterator_sel_t type)
gpi_sim_hdl gpi_next (gpi_iterator_hdl iterator)
int gpi_get_num_elems (gpi_sim_hdl gpi_sim_hdl)
int gpi_get_range_left (gpi_sim_hdl gpi_sim_hdl)
int gpi_get_range_right (gpi_sim_hdl gpi_sim_hdl)
const char *gpi_get_signal_value_binstr (gpi_sim_hdl gpi_hdl)
const char *gpi_get_signal_value_str (gpi_sim_hdl gpi_hdl)
double gpi_get_signal_value_real (gpi_sim_hdl gpi_hdl)
long gpi_get_signal_value_long (gpi_sim_hdl gpi_hdl)
const char *gpi_get_signal_name_str (gpi_sim_hdl gpi_hdl)
const char *gpi_get_signal_type_str (gpi_sim_hdl gpi_hdl)
gpi_objtype_t gpi_get_object_type (gpi_sim_hdl gpi_hdl)
const char *gpi_get_definition_name (gpi_sim_hdl gpi_hdl)
const char *gpi_get_definition_file (gpi_sim_hdl gpi_hdl)
int gpi_is_constant (gpi_sim_hdl gpi_hdl)
int gpi_is_indexable (gpi_sim_hdl gpi_hdl)
void gpi_set_signal_value_real (gpi_sim_hdl gpi_hdl, double value)
void gpi_set_signal_value_long (gpi_sim_hdl gpi_hdl, long value)
void gpi_set_signal_value_str (gpi_sim_hdl gpi_hdl, const char *str)
gpi_sim_hdl gpi_register_timed_callback (int (*gpi_function)) const void *
  , void *gpi_cb_data, uint64_t time_ps

gpi_sim_hdl gpi_register_value_change_callback (int (*gpi_function)) const void *
  , void *gpi_cb_data, gpi_sim_hdl gpi_hdl, unsigned int edge

gpi_sim_hdl gpi_register_readonly_callback (int (*gpi_function)) const void *
  , void *gpi_cb_data

gpi_sim_hdl gpi_register_nexttime_callback (int (*gpi_function)) const void *
  , void *gpi_cb_data


```c
void gpi_deregister_callback(gpi_sim_hdl gpi_hdl)

void *gpi_get_callback_data(gpi_sim_hdl gpi_hdl)

int gpi_print_registered_impl(void)
```

```c
struct gpi_sim_info_s
```

**Public Members**

```c
int32_t argc
char **argv
char *product
char *version
int32_t *reserved[4]
```

**File gpi_embed.c**

**Initialize the Python interpreter**

Create and initialize the Python interpreter.

GILState before calling: N/A

GILState after calling: released

Stores the thread state for cocotb in static variable gtstate

void `embed_init_python` (void)

**Simulator cleanup**

Called by the simulator on shutdown.

GILState before calling: Not held

GILState after calling: Not held

Makes one call to PyGILState_Ensure and one call to Py_Finalize.

Cleans up reference counts for Python objects and calls Py_Finalize function.

void `embed_sim_cleanup` (void)
Initialization

Called by the simulator on initialization.

Load cocotb Python module

GILState before calling: Not held

GILState after calling: Not held

Makes one call to PyGILState_Ensure and one call to PyGILState_Release

Loads the Python module called cocotb and calls the _initialise_testbench function

```c
int get_module_ref (const char *modname, PyObject **mod)
int embed_sim_init (gpi_sim_info_t *info)
void embed_sim_event (gpi_event_t level, const char *msg)
```

Functions

```c
static void set_program_name_in_venv (void)
```

Variables

```c
PyThreadState *gtstate = NULL
char progname[] = "cocotb"
char *argv[] = {progname}
const char *PYTHON_INTERPRETER_PATH = "/bin/python"
PyObject *pEventFn = NULL
```

File gpi_logging.c

GPI logging

Write a log message using cocotb SimLog class

GILState before calling: Unknown

GILState after calling: Unknown

Makes one call to PyGILState_Ensure and one call to PyGILState_Release

If the Python logging mechanism is not initialised, dumps to stderr.

```c
void gpi_log (const char *name, long level, const char *pathname, const char *funcname, long lineno, const char *msg, ...)
```
Defines

LOG_SIZE 512

Functions

void set_log_handler (void *handler)
void clear_log_handler (void)
void set_log_filter (void *filter)
void clear_log_filter (void)
void set_log_level (enum gpi_log_levels new_level)
const char *log_level (long level)

Variables

PyObject *pLogHandler = NULL
PyObject *pLogFilter = NULL
gpi_log_levels local_level = GPIInfo

struct _log_level_table log_level_table[] = {
    { 10, "DEBUG" },
    { 20,

char log_buff[LOG_SIZE]

struct _log_level_table

Public Members

long level

const char *levelname

File gpi_logging.h

Defines

EXTERN_C_START
EXTERN_C_END

LOG_DEBUG (...) gpi_log("cocotb.gpi", GPIDebug, _FILE__, _func__, _LINE__, _VA_ARGS__);
LOG_INFO (...) gpi_log("cocotb.gpi", GPIInfo, _FILE__, _func__, _LINE__, _VA_ARGS__);
LOG_WARN (...) gpi_log("cocotb.gpi", GPIWarning, _FILE__, _func__, _LINE__, _VA_ARGS__);
LOG_ERROR (...) gpi_log("cocotb.gpi", GPIError, _FILE__, _func__, _LINE__, _VA_ARGS__);
        gpi_log("cocotb.gpi", GPCritical, _FILE__, _func__, _LINE__, _VA_ARGS__);\ exit(1); \} while (0) 
FENTER
FEXIT
Enums

`enum gpi_log_levels
    Values:
    GPIDebug = 10
    GPIInfo = 20
    GPIWarning = 30
    GPIError = 40
    GPICritical = 50`

Functions

`void set_log_handler (void *handler)`
`void clear_log_handler (void)`
`void set_make_record (void *makerecord)`
`void set_log_filter (void *filter)`
`void clear_log_filter (void)`
`void set_log_level (enum gpi_log_levels new_level)`
`void gpi_log (const char *name, long level, const char *pathname, const char *funcname, long lineno, const char *msg, ...)`

File `gpi_priv.h`

Defines

`const void NAME##_entry_point() { func(); } }`

Typedefs

`typedef enum gpi_cb_state gpi_cb_state_e`
`typedef const void (*layer_entry_func)()`

Enums

`enum gpi_cb_state
    Values:
    GPI_FREE = 0
    GPI_PRIMED = 1
    GPI_CALL = 2
    GPI_REPRIME = 3
    GPI_DELETE = 4`
Functions

template<class To>
To sim_to_hdl (gpi_sim_hdl input)

int gpi_register_impl (GpimplInterface *func_tbl)

void gpi_embed_init (gpi_sim_info_t *info)
void gpi_cleanup ()
void gpi_embed_end ()
void gpi_embed_event (gpi_event_t level, const char *msg)
void gpi_load_extra_libs ()

class GpiHdl
  Subclassed by GpiCbHdl, Gpilterator, GpiObjHdl

Public Functions

GpiHdl (GpimplInterface *impl)
GpiHdl (GpimplInterface *impl, void *hdl)
virtual ~GpiHdl ()

int initialise (std::string &name)

template<typename T>
T get_handle () const

char *gpi_copy_name (const char *name)

bool is_this_impl (GpimplInterface *impl)

Public Members

GpimplInterface *m_impl

Protected Attributes

void *m_obj_hdl

Private Functions

GpiHdl ()

class GpiObjHdl : public GpiHdl
  Subclassed by FliObjHdl, GpiSignalObjHdl, VhpiArrayObjHdl, VhpiObjHdl, VpiArrayObjHdl, VpiObjHdl

8.1. API Documentation
Public Functions

GpiObjHdl (GpiImplInterface *impl)
GpiObjHdl (GpiImplInterface *impl, void *hdl, gpi_objtype_t objtype)
GpiObjHdl (GpiImplInterface *impl, void *hdl, gpi_objtype_t objtype, bool is_const)
virtual ~GpiObjHdl ()
const char *get_name_str ()
const char *get_fullname_str ()
const char *get_type_str ()
gpi_objtype_t get_type ()
bool get_const ()
int get_num_elems ()
int get_range_left ()
int get_range_right ()
int get_indexable ()
const std::string &get_name ()
const std::string &get_fullname ()
virtual const char *get_definition_name ()
virtual const char *get_definition_file ()
bool is_native_impl (GpiImplInterface *impl)
int initialise (std::string &name, std::string &full_name)

Protected Attributes

int m_num_elems
bool m_indexable
int m_range_left
int m_range_right
std::string m_name
std::string m_fullname
std::string m_definition_name
std::string m_definition_file
gpi_objtype_t m_type
bool m_const

class GpiSignalObjHdl : public GpiObjHdl
   Subclassed by FliSignalObjHdl, VhpiSignalObjHdl, VpiSignalObjHdl
Public Functions

GpiSignalObjHdl (GpiImplInterface *impl, void *hdl, gpi_objtype_t objtype, bool is_const)
virtual ~GpiSignalObjHdl()
virtual const char *get_signal_value_binstr () = 0
virtual const char *get_signal_value_str () = 0
virtual double get_signal_value_real () = 0
virtual long get_signal_value_long () = 0
virtual int set_signal_value (const long value) = 0
virtual int set_signal_value (const double value) = 0
virtual int set_signal_value (std::string &value) = 0
virtual GpiCbHdl *value_change_cb (unsigned int edge) = 0

Public Members

int m_length

class GpiCbHdl : public GpiHdl
Subclassed by FliProcessCbHdl, GpiValueCbHdl, VhpiCbHdl, VpiCbHdl

Public Functions

GpiCbHdl (GpiImplInterface *impl)
int arm_callback () = 0
int run_callback ()
int cleanup_callback () = 0
int set_user_data (int (*gpi_function)) const void *
   , const void *data
const void *get_user_data ()
void set_call_state (gpi_cb_state_e new_state)
gpi_cb_state_e get_call_state ()
~GpiCbHdl ()
Protected Attributes

int (*gpi_function)(const void *)
const void *m_cb_data
gpi_cb_state *m_state

class GpiValueCbHdl : public virtual GpiCbHdl
Subclassed by FliSignalCbHdl, VhpiValueCbHdl, VpiValueCbHdl

Public Functions

GpiValueCbHdl(GpiImplInterface *impl, GpiSignalObjHdl *signal, int edge)
virtual ~GpiValueCbHdl()
int run_callback()
virtual int cleanup_callback() = 0

Protected Attributes

std::string required_value
GpiSignalObjHdl *m_signal

class GpiClockHdl

Public Functions

GpiClockHdl(GpiObjHdl *clk)
GpiClockHdl(const char *clk)
~GpiClockHdl()
int start_clock(const int period_ps)
int stop_clock()

class GpiIterator : public GpiHdl
Subclassed by FliIterator, VhpiIterator, VpiIterator, VpiSingleIterator

Public Types

enum Status
Values:
    NATIVE
    NATIVE_NO_NAME
    NOT_NATIVE
    NOT_NATIVE_NO_NAME
    END
Public Functions

GpiIterator (GpiImplInterface *impl, GpiObjHdl *hdl)
virtual ~GpiIterator ()
virtual Status next_handle (std::string &name, GpiObjHdl **hdl, void **raw_hdl)
GpiObjHdl *get_parent ()

Protected Attributes

GpiObjHdl *m_parent

template<class Ti, class Tm>
class GpiIteratorMapping

Public Functions

GpiIteratorMapping (void (*populate)) GpiIteratorMapping<Ti, Tm> &
std::vector<Tm> *get_options (Ti type)
void add_to_options (Ti type, Tm *options)

Private Members

std::map<Ti, std::vector<Tm>> options_map

class GpiImplInterface
Subclassed by FiiImpl, VhpiImpl, VpiImpl

Public Functions

GpiImplInterface (const std::string &name)
const char *get_name_c ()
const string &get_name_s ()
virtual ~GpiImplInterface ()
virtual void sim_end () = 0
virtual void get_sim_time (uint32_t *high, uint32_t *low) = 0
virtual void get_sim_precision (int32_t *precision) = 0
virtual GpiObjHdl *native_check_create (std::string &name, GpiObjHdl *parent) = 0
virtual GpiObjHdl *native_check_create (int32_t index, GpiObjHdl *parent) = 0
virtual GpiObjHdl *native_check_create (void *raw_hdl, GpiObjHdl *parent) = 0
virtual GpiObjHdl *get_root_handle (const char *name) = 0
virtual GpiIterator *iterate_handle (GpiObjHdl *obj_hdl, gpi_iterator_sel_t type) = 0

virtual GpiCbHdl *register_timed_callback (uint64_t time_ps) = 0

virtual GpiCbHdl *register_readonly_callback () = 0

virtual GpiCbHdl *register_nexttime_callback () = 0

virtual GpiCbHdl *register_readwrite_callback () = 0

virtual int deregister_callback (GpiCbHdl *obj_hdl) = 0

virtual const char *reason_to_string (int reason) = 0

**Private Members**

std::string m_name

**File python3_compat.h**

**Defines**

GETSTATE (m) (&_state)

MODULE_ENTRY_POINT initsimulator

INITERROR return

struct module_state

**Public Members**

PyObject *error

**File simulatormodule.c**

Python extension to provide access to the simulator.
Uses GPI calls to interface to the simulator.

**Callback Handling**

Handle a callback coming from GPI
GILState before calling: Unknown
GILState after calling: Unknown
Makes one call to TAKE_GIL and one call to DROP_GIL
Returns 0 on success or 1 on a failure.
Handles a callback from the simulator, all of which call this function.
We extract the associated context and find the Python function (usually cocotb.scheduler.react) calling it with a reference to the trigger that fired. The scheduler can then call next() on all the coroutines that are waiting on that particular trigger.
TODO:

- Tidy up return values
- Ensure cleanup correctly in exception cases

```c
int handle_gpi_callback (void *user_data)

static PyObject *log_msg (PyObject *self, PyObject *args)
static PyObject *register_readonly_callback (PyObject *self, PyObject *args)
static PyObject *register_rwsynch_callback (PyObject *self, PyObject *args)
static PyObject *register_nextstep_callback (PyObject *self, PyObject *args)
static PyObject *register_timed_callback (PyObject *self, PyObject *args)
static PyObject *register_value_change_callback (PyObject *self, PyObject *args)
static PyObject *iterate (PyObject *self, PyObject *args)
static PyObject *next (PyObject *self, PyObject *args)
static PyObject *get_signal_val_binstr (PyObject *self, PyObject *args)
static PyObject *get_signal_val_str (PyObject *self, PyObject *args)
static PyObject *get_signal_val_real (PyObject *self, PyObject *args)
static PyObject *get_signal_val_long (PyObject *self, PyObject *args)
static PyObject *set_signal_val_str (PyObject *self, PyObject *args)
static PyObject *set_signal_val_real (PyObject *self, PyObject *args)
static PyObject *set_signal_val_long (PyObject *self, PyObject *args)
static PyObject *get_definition_name (PyObject *self, PyObject *args)
static PyObject *get_definition_file (PyObject *self, PyObject *args)
static PyObject *get_handle_by_name (PyObject *self, PyObject *args)
static PyObject *get_handle_by_index (PyObject *self, PyObject *args)
static PyObject *get_root_handle (PyObject *self, PyObject *args)
static PyObject *get_name_string (PyObject *self, PyObject *args)
static PyObject *get_type (PyObject *self, PyObject *args)
static PyObject *get_const (PyObject *self, PyObject *args)
static PyObject *get_type_string (PyObject *self, PyObject *args)
static PyObject *get_sim_time (PyObject *self, PyObject *args)
static PyObject *get_precision (PyObject *self, PyObject *args)
static PyObject *get_num_elems (PyObject *self, PyObject *args)
static PyObject *get_range (PyObject *self, PyObject *args)
static PyObject *stop_simulator (PyObject *self, PyObject *args)
static PyObject *deregister_callback (PyObject *self, PyObject *args)
static PyObject *log_level (PyObject *self, PyObject *args)
static void add_module_constants (PyObject *simulator)
```
Typedefs

typedef int (*gpi_function_t)(const void *)

Functions

PyGILState_STATE TAKE_GIL (void)
void DROP_GIL (PyGILState_STATE state)
static int gpi_sim_hdl_converter (PyObject *, gpi_sim_hdl *)
static int gpi_iterator_hdl_converter (PyObject *, gpi_iterator_hdl *)

Variables

int takes = 0
int releases = 0
int sim_ending = 0
struct sim_time cache_time
struct sim_time

Public Members

uint32_t high
uint32_t low

File simulatormodule.h

Defines

COCOTB_ACTIVE_ID 0xC0C07B
COCOTB_INACTIVE_ID 0xDEADB175
MODULE_NAME "simulator"

Typedefs

typedef struct t_callback_data s_callback_data
typedef struct t_callback_data *p_callback_data
Functions

static PyObject *error_out (PyObject *m)
static PyObject *log_msg (PyObject *self, PyObject *args)
static PyObject *get_signal_val_long (PyObject *self, PyObject *args)
static PyObject *get_signal_val_real (PyObject *self, PyObject *args)
static PyObject *get_signal_val_str (PyObject *self, PyObject *args)
static PyObject *get_signal_val_binstr (PyObject *self, PyObject *args)
static PyObject *set_signal_val_long (PyObject *self, PyObject *args)
static PyObject *set_signal_val_real (PyObject *self, PyObject *args)
static PyObject *set_signal_val_str (PyObject *self, PyObject *args)
static PyObject *get_definition_name (PyObject *self, PyObject *args)
static PyObject *get_definition_file (PyObject *self, PyObject *args)
static PyObject *get_handle_by_name (PyObject *self, PyObject *args)
static PyObject *get_handle_by_index (PyObject *self, PyObject *args)
static PyObject *get_root_handle (PyObject *self, PyObject *args)
static PyObject *get_name_string (PyObject *self, PyObject *args)
static PyObject *get_type (PyObject *self, PyObject *args)
static PyObject *get_const (PyObject *self, PyObject *args)
static PyObject *get_type_string (PyObject *self, PyObject *args)
static PyObject *get_num_elems (PyObject *self, PyObject *args)
static PyObject *get_range (PyObject *self, PyObject *args)
static PyObject *register_timed_callback (PyObject *self, PyObject *args)
static PyObject *register_value_change_callback (PyObject *self, PyObject *args)
static PyObject *register_readonly_callback (PyObject *self, PyObject *args)
static PyObject *register_nextstep_callback (PyObject *self, PyObject *args)
static PyObject *register_rwsynch_callback (PyObject *self, PyObject *args)
static PyObject *stop_simulator (PyObject *self, PyObject *args)
static PyObject *iterate (PyObject *self, PyObject *args)
static PyObject *next (PyObject *self, PyObject *args)
static PyObject *get_sim_time (PyObject *self, PyObject *args)
static PyObject *get_precision (PyObject *self, PyObject *args)
static PyObject *deregister_callback (PyObject *self, PyObject *args)
static PyObject *log_level (PyObject *self, PyObject *args)
Variables

PyMethodDef SimulatorMethods[]
struct t_callback_data

Public Members

PyThreadState * _saved_thread_state
uint32_t tid_value
PyObject * function
PyObject * args
PyObject * kwargs
      gpi_sim_hdl cb_hdl

File simulatormodule_python2.c

Functions

static PyObject * error_out (PyObject * m)
PyMODINIT_FUNC MODULE_ENTRY_POINT (void)

Variables

char error_module[] = MODULE_NAME ".Error"
struct module_state _state

File simulatormodule_python3.c

Functions

static PyObject * error_out (PyObject * m)
static int simulator_traverse (PyObject * m, visitproc visit, void * arg)
static int simulator_clear (PyObject * m)
PyMODINIT_FUNC MODULE_ENTRY_POINT (void)

Variables

struct PyModuleDef moduledef = {PyModuleDef_HEAD_INIT, , , , , , , , }

108 Chapter 8. C Code Library Reference
File sv_vpi_user.h

Defines

vpiPackage 600
vpiInterface 601
vpiProgram 602
vpiInterfaceArray 603
vpiProgramArray 604
vpiTypespec 605
vpiModport 606
vpiInterfaceTfDecl 607
vpiRefObj 608
vpiTypeParameter 609
vpiVarBit vpiRegBit
vpiLongIntVar 610
vpiShortIntVar 611
vpiIntVar 612
vpiShortRealVar 613
vpiByteVar 614
vpiClassVar 615
vpiStringVar 616
vpiEnumVar 617
vpiStructVar 618
vpiUnionVar 619
vpiBitVar 620
vpiLogicVar vpiReg
vpiArrayVar vpiRegArray
vpiClassObj 621
vpiCHandleVar 622
vpiPackedArrayVar 623
vpiVirtualInterfaceVar 728
vpiLongIntTypespec 625
vpiShortRealTypespec 626
vpiByteTypespec 627
vpiShortIntTypespec 628
vpiIntTypespec 629
vpiClassTypespec 630
vpiStringTypespec 631
vpiCHandleTypespec 632
vpiEnumTypespec 633
vpiEnumConst 634
vpiIntegerTypespec 635
vpiTimeTypespec 636
vpiRealTypespec 637
vpiStructTypespec 638
vpiUnionTypespec 639
vpiBitTypespec 640
vpiLogicTypespec 641
vpiArrayTypespec 642
vpiVoidTypespec 643
vpiTypespecMember 644
vpiPackedArrayTypespec 692
vpiSequenceTypespec 696
vpiPropertyTypespec 697
vpiEventTypespec 698
vpiClockingBlock 650
vpiClockingIODevice 651
vpiClassDefn 652
vpiConstraint 653
vpiConstraintOrdering 654
vpiDistItem 645
vpiAliasStmt 646
vpiThread 647
vpiMethodFuncCall 648
vpiMethodTaskCall 649
vpiAssert 686
vpiAssume 687
vpiCover 688
vpiRestrict 901
vpiDisableCondition 689
vpiClockingEvent 690
vpiPropertyDecl 655
8.1. API Documentation
vpiContinue 685
vpiPackedArrayNet 693
vpiConstraintExpr 747
vpiElseConst 748
vpiImplication 749
vpiConstrIf 738
vpiConstrIfElse 739
vpiConstrForEach 736
vpiLetDecl 903
vpiLetExpr 904
vpiActual 700
vpiTypedefAlias 701
vpiIndexTypespec 702
vpiBaseTypespec 703
vpiElemTypespec 704
vpiInputSkew 706
vpiOutputSkew 707
vpiGlobalClocking 708
vpiDefaultClocking 709
vpiDefaultDisableIff 710
vpiOrigin 713
vpiPrefix 714
vpiWith 715
vpiProperty 718
vpiValueRange 720
vpiPattern 721
vpiWeight 722
vpiConstraintItem 746
vpiTypedef 725
vpiImport 726
vpiDerivedClasses 727
vpiInterfaceDecl vpiVirtualInterfaceVar /**< interface decl deprecated */
vpiMethods 730
vpiSolveBefore 731
vpiSolveAfter 732
vpiWaitingProcesses 734
vpiMessages 735
vpiLoopVars 737
vpiConcurrentAssertions 740
vpiMatchItem 741
vpiMember 742
vpiElement 743
vpiAssertion 744
vpiInstance 745
vpiTop 600
vpiUnit 602
vpiJoinType 603
vpiJoin 0
vpiJoinNone 1
vpiJoinAny 2
vpiAccessType 604
vpiForkJoinAcc 1
vpiExternAcc 2
vpiDPIExportAcc 3
vpiDPIImportAcc 4
vpiArrayType 606
vpiStaticArray 1
vpiDynamicArray 2
vpiAssocArray 3
vpiQueueArray 4
vpiArrayMember 607
vpiIsRandomized 608
vpiLocalVarDecl 609
vpiOpStrong 656 /* strength of temporal operator */
vpiRandType 610
vpiNotRand 1
vpiRand 2
vpiRandC 3
vpiPortType 611
vpiInterfacePort 1
vpiModportPort 2
vpiConstantVariable 612

8.1. API Documentation 113
vpiStructUnionMember 615
vpiVisibility 620
vpiPublicVis 1
vpiProtectedVis 2
vpiLocalVis 3
vpiOneStepConst 9
vpiUnboundedConst 10
vpiNullConst 11
vpiAlwaysType 624
vpiAlwaysComb 2
vpiAlwaysFF 3
vpiAlwaysLatch 4
vpiDistType 625
vpiEqualDist 1 /* constraint equal distribution */
vpiDivDist 2 /* constraint divided distribution */
vpiPacked 630
vpiTagged 632
vpiRef 6 /* Return value for vpiDirection property */
vpiVirtual 635
vpiHasActual 636
vpiIsConstraintEnabled 638
vpiSoft 639
vpiClassType 640
vpiMailboxClass 1
vpiSemaphoreClass 2
vpiUserDefinedClass 3
vpiProcessClass 4
vpiMethod 645
vpiIsClockInferred 649
vpiIsDeferred 657
vpiIsFinal 658
vpiIsCoverSequence 659
vpiQualifier 650
vpiNoQualifier 0
vpiUniqueQualifier 1
vpiPriorityQualifier 2
vpiTaggedQualifier 4
vpiRandQualifier 8
vpiInsideQualifier 16
  vpiNedge */
  vpiNedge */
vpiGeneric 653
vpiCompatibilityMode 654
vpiModel1364v1995 1
vpiModel1364v2001 2
vpiModel1364v2005 3
vpiModel1800v2005 4
vpiModel1800v2009 5
vpiPackedArrayMember 655
vpiStartLine 661
vpiColumn 662
vpiEndLine 663
vpiEndColumn 664
vpiAllocScheme 658
vpiAutomaticScheme 1
vpiDynamicScheme 2
vpiOtherScheme 3
vpiObjId 660
vpiDPIPure 665
vpiDPIContext 666
vpiDPIStr 667
vpiDPI 1
vpiDPI 2
vpiDPIIdentifier 668
vpiImplyOp 50 /* -> implication operator */
vpiNonOverlapImplyOp 51 /* |=> nonoverlapped implication */
vpiOverlapImplyOp 52 /* |-> overlapped implication operator */
vpiAcceptOnOp 83 /* accept_on operator */
vpiRejectOnOp 84 /* reject_on operator */
vpiSyncAcceptOnOp 85 /* sync_accept_on operator */
vpiSyncRejectOnOp 86 /* sync_reject_on operator */
vpiOverlapFollowedByOp 87 /* overlapped followed_by operator */
vpiNonOverlapFollowedByOp 88 /* nonoverlapped followed_by operator */
vpiNexttimeOp 89 /* nexttime operator */
vpiAlwaysOp 90 /* always operator */
vpiEventuallyOp 91 /* eventually operator */
vpiUntilOp 92 /* until operator */
vpiUntilWithOp 93 /* until_with operator */
vpiUnaryCycleDelayOp 53 /* binary cycle delay (##) operator */
vpiCycleDelayOp 54 /* binary cycle delay (##) operator */
vpiIntersectOp 55 /* intersection operator */
vpiFirstMatchOp 56 /* first_match operator */
vpiThroughoutOp 57 /* throughout operator */
vpiWithinOp 58 /* within operator */
vpiRepeatOp 59 /* [=] nonconsecutive repetition */
vpiConsecutiveRepeatOp 60 /* [*] consecutive repetition */
vpiGotoRepeatOp 61 /* [->] goto repetition */
vpiPostIncOp 62 /* ++ post-increment */
vpiPreIncOp 63 /* ++ pre-increment */
vpiPostDecOp 64 /* – post-decrement */
vpiPreDecOp 65 /* – pre-decrement */
vpiMatchOp 66 /* match() operator */
vpiCastOp 67 /* type() operator */
vpiIffOp 68 /* iff operator */
vpiWildEqOp 69 /* ==? operator */
vpiWildNeqOp 70 /* !=? operator */
vpiStreamLROp 71 /* left-to-right streaming {>>} operator */
vpiStreamRLOp 72 /* right-to-left streaming {<<} operator */
vpiMatchedOp 73 /* the .matched sequence operation */
vpiTriggeredOp 74 /* the .triggered sequence operation */
vpiAssignmentPatternOp 75 /* {} assignment pattern */
vpiMultiAssignmentPatternOp 76 /* {n{}} multi assignment pattern */
vpiIfOp 77 /* if operator */
vpiIfElseOp 78 /* if/else operator */
vpiCompAndOp 79 /* Composite and operator */
vpiCompOrOp 80 /* Composite or operator */
vpiImpliesOp 94 /* implies operator */
vpiInsideOp 95 /* inside operator */
vpiTypeOp 81 /* type operator */
vpiAssignmentOp 82 /* Normal assignment */
vpiOtherFunc 6 /* returns other types; for property vpiFuncType */
vpiValidUnknown 2 /* Validity of variable is unknown */
cbStartOfThread 600 /* callback on thread creation */
cbEndOfThread 601 /* callback on thread termination */
cbEnterThread 602 /* callback on reentering thread */
cbStartOfFrame 603 /* callback on frame creation */
cbEndOfFrame 604 /* callback on frame exit */
cbSizeChange 605 /* callback on array variable size change */
cbCreateObj 700 /* callback on class object creation */
cbReclaimObj 701 /* callback on class object reclaimed by automatic memory management */
cbEndOfClassObject 702 /* callback on transient object deletion */
vpiCoverageStart 750
vpiCoverageStop 751
vpiCoverageReset 752
vpiCoverageCheck 753
vpiCoverageMerge 754
vpiCoverageSave 755
vpiAssertCoverage 760
vpiFsmStateCoverage 761
vpiStatementCoverage 762
vpiToggleCoverage 763
vpiCovered 765
vpiCoverMax 766
vpiCoveredCount 767
vpiAssertAttemptCovered 770
vpiAssertSuccessCovered 771
vpiAssertFailureCovered 772
vpiAssertVacuousSuccessCovered 773
vpiAssertDisableCovered 774
vpiAssertKillCovered 777
vpiFsmStates 775
vpiFsmStateExpression 776
vpiFsm 758
vpiFsmHandle 759
cbAssertionStart 606
cbAssertionSuccess 607
cbAssertionFailure 608
cbAssertionVacuousSuccess 657
cbAssertionDisabledEvaluation 658
cbAssertionStepSuccess 609
cbAssertionStepFailure 610
cbAssertionLock 661
cbAssertionUnlock 662
cbAssertionDisable 611
cbAssertionEnable 612
cbAssertionReset 613
cbAssertionKill 614
cbAssertionEnablePassAction 645
cbAssertionEnableFailAction 646
cbAssertionDisablePassAction 647
cbAssertionDisableFailAction 648
cbAssertionEnableNonvacuousAction 649
cbAssertionDisableVacuousAction 650
cbAssertionSysInitialized 615
cbAssertionSysOn 616
cbAssertionSysOff 617
cbAssertionSysKill 631
cbAssertionSysLock 659
cbAssertionSysUnlock 660
cbAssertionSysEnd 618
cbAssertionSysReset 619
cbAssertionSysEnablePassAction 651
cbAssertionSysEnableFailAction 652
cbAssertionSysDisablePassAction 653
cbAssertionSysDisableFailAction 654
cbAssertionSysEnableNonvacuousAction 655
cbAssertionSysDisableVacuousAction 656
vpiAssertionLock 645
vpiAssertionUnlock 646
vpiAssertionDisable 620
vpiAssertionEnable 621
vpiAssertionReset 622
vpiAssertionKill 623
vpiAssertionEnableStep 624
vpiAssertionDisableStep 625
vpiAssertionClockSteps 626
vpiAssertionSysLock 647
vpiAssertionSysUnlock 648
vpiAssertionSysOn 627
vpiAssertionSysOff 628
vpiAssertionSysKill 632
vpiAssertionSysEnd 629
vpiAssertionSysReset 630
vpiAssertionDisablePassAction 633
vpiAssertionEnablePassAction 634
vpiAssertionDisableFailAction 635
vpiAssertionEnableFailAction 636
vpiAssertionDisableVacuousAction 637
vpiAssertionEnableNonvacuousAction 638
vpiAssertionSysEnablePassAction 639
vpiAssertionSysEnableFailAction 640
vpiAssertionSysDisablePassAction 641
vpiAssertionSysDisableFailAction 642
vpiAssertionSysEnableNonvacuousAction 643
vpiAssertionSysDisableVacuousAction 644

**Typedefs**

typedef struct t_vpi_assertion_step_info s_vpi_assertion_step_info
typedef struct t_vpi_assertion_step_info *p_vpi_assertion_step_info
typedef struct t_vpi_attempt_info s_vpi_attempt_info
typedef struct t_vpi_attempt_info *p_vpi_attempt_info
typedef PLI_INT32() vpi_assertion_callback_func(PLI_INT32 reason, p_vpi_time cb_time, vpiHandle)
Functions

vpiHandle vpi_register_assertion_cb (vpiHandle assertion, vpi_assertion_callback_func *cb_rtn, PLI_BYTE8 *user_data)

struct t_vpi_assertion_step_info

Public Members

PLI_INT32 matched_expression_count
vpiHandle *matched_exprs
PLI_INT32 stateFrom
PLI_INT32 stateTo

struct t_vpi_attempt_info

Public Members

vpiHandle failExpr
p_vpi_assertion_step_info step
union t_vpi_attempt_info::[anonymous] detail
s_vpi_time attemptStartTime

File verilator.cpp

Functions

double sc_time_stamp ()

void vlog_startup_routines_bootstrap (void)

int main (int argc, char **argv)

Variables

vluint64_t main_time = 0

File vhpi_user.h

Defines

PLI_DLLISPEC
PLI_DLLLESPEC
PLI_EXTERN
PLI_VEXTERN
PLI_PROTOTYPES
EXTERN PLI_EXTERN PLI_DLLSPEC
EXTERN PLI_EXTERN PLI_DLLESPEC
VHPI_TYPES
PLI_TYPES
vhpiUndefined -1
vhpiU 0 /* uninitialized */
vhpiX 1 /* unknown */
vhpi0 2 /* forcing 0 */
vhpi1 3 /* forcing 1 */
vhpiZ 4 /* high impedance */
vhpiW 5 /* weak unknown */
vhpiL 6 /* weak 0 */
vhpiH 7 /* weak 1 */
vhpiDontCare 8 /* don’t care */
vhpibit0 0 /* bit 0 */
vhpibit1 1 /* bit 1 */
vhpiFalse 0 /* false */
vhpiTrue 1 /* true */
vhpiCbValueChange 1001
vhpiCbForce 1002
vhpiCbRelease 1003
vhpiCbTransaction 1004 /* optional callback reason */
vhpiCbStmt 1005
vhpiCbResume 1006
vhpiCbSuspend 1007
vhpiCbStartOfSubpCall 1008
vhpiCbEndOfSubpCall 1009
vhpiCbAfterDelay 1010
vhpiCbRepAfterDelay 1011
vhpiCbNextTimeStep 1012
vhpiCbRepNextTimeStep 1013
vhpiCbStartOfNextCycle 1014
vhpiCbRepStartOfNextCycle 1015
vhpiCbStartOfProcesses 1016
vhpiCbRepStartOfProcesses 1017
vhpiCbEndOfProcesses 1018
vhpiCbRepEndOfProcesses 1019
vhpiCbLastKnownDeltaCycle 1020
vhpiCbRepLastKnownDeltaCycle 1021
vhpiCbStartOfPostponed 1022
vhpiCbRepStartOfPostponed 1023
vhpiCbEndOfTimeStep 1024
vhpiCbRepEndOfTimeStep 1025
vhpiCbStartOfTool 1026
vhpiCbEndOfTool 1027
vhpiCbStartOfAnalysis 1028
vhpiCbEndOfAnalysis 1029
vhpiCbStartOfElaboration 1030
vhpiCbEndOfElaboration 1031
vhpiCbStartOfInitialization 1032
vhpiCbEndOfInitialization 1033
vhpiCbStartOfSimulation 1034
vhpiCbEndOfSimulation 1035
vhpiCbQuiescense 1036 /* repetitive */
vhpiCbPLIError 1037 /* repetitive */
vhpiCbStartOfSave 1038
vhpiCbEndOfSave 1039
vhpiCbStartOfRestart 1040
vhpiCbEndOfRestart 1041
vhpiCbStartOfReset 1042
vhpiCbEndOfReset 1043
vhpiCbEnterInteractive 1044 /* repetitive */
vhpiCbExitInteractive 1045 /* repetitive */
vhpiCbSigInterrupt 1046 /* repetitive */
vhpiCbTimeOut 1047 /* non repetitive */
vhpiCbRepTimeOut 1048 /* repetitive */
vhpiCbSensitivity 1049 /* repetitive */
vhpiReturnCb 0x00000001
vhpiDisableCb 0x00000010
VHPI_SENS_ZERO (sens) vhpi_sens_zero(sens)
VHPI_SENS_SET (obj, sens) vhpi_sens_set(obj, sens)
VHPI_SENS_CLR (obj, sens) vhpi_sens_clr(obj, sens)
VHPI_SENS_ISSET (obj, sens) vhpi_sens_isset(obj, sens)
VHPI_SENS_FIRST (sens) vhpi_sens_first(sens)
vhpiNoActivity -1

Typedefs

typedef uint32_t *vhpiHandleT
typedef uint32_t vhpiEnumT
typedef uint8_t vhpiSmallEnumT
typedef uint32_t vhpiIntT
typedef uint64_t vhpiLongIntT
typedef char vhpiCharT
typedef double vhpiRealT
typedef uint32_t vhpiSmallPhysT
typedef struct vhpiPhysS vhpiPhysT
typedef int PLI_INT32
typedef unsigned int PLI_UINT32
typedef short PLI_INT16
typedef unsigned short PLI_UINT16
typedef char PLI_BYTE8
typedef unsigned char PLI_UBYTE8
typedef void PLI_VOID
typedef struct vhpiTimeS vhpiTimeT
typedef struct vhpiValueS vhpiValueT
typedef struct vhpiErrorInfoS vhpiErrorInfoT
typedef struct vhpiCbDataS vhpiCbDataT
typedef int (*vhpiUserFctT) (void)
typedef struct vhpiForeignDataS vhpiForeignDataT
typedef void (*vhpiBootstrapFctT) (void)

Enums

enum vhpiFormatT
   Values:
      vhpiBinStrVal = 1
      vhpiOctStrVal = 2
      vhpiDecStrVal = 3
      vhpiHexStrVal = 4
vhpiEnumVal = 5
vhpiIntVal = 6
vhpiLogicVal = 7
vhpiRealVal = 8
vhpiStrVal = 9
vhpiCharVal = 10
vhpiTimeVal = 11
vhpiPhysVal = 12
vhpiObjectTypeVal = 13
vhpiPtrVal = 14
vhpiEnumVecVal = 15
vhpiIntVecVal = 16
vhpiLogicVecVal = 17
vhpiRealVecVal = 18
vhpiTimeVecVal = 19
vhpiPhysVecVal = 20
vhpiPtrVecVal = 21
vhpiRawDataVal = 22
vhpiSmallEnumVal = 23
vhpiSmallEnumVecVal = 24
vhpiLongIntVal = 25
vhpiLongIntVecVal = 26
vhpiSmallPhysVal = 27
vhpiSmallPhysVecVal = 28

enum vhpiClassKindT
Values:
  vhpiAccessTypeDeclK = 1001
  vhpiAggregateK = 1002
  vhpiAliasDeclK = 1003
  vhpiAllK = 1004
  vhpiAllocatorK = 1005
  vhpiAnyCollectionK = 1006
  vhpiArchBodyK = 1007
  vhpiArgvK = 1008
  vhpiArrayTypeDeclK = 1009
  vhpiAssertStmtK = 1010
vhpiAssocElemK = 1011
vhpiAttrDeclK = 1012
vhpiAttrSpecK = 1013
vhpiBinaryExprK = 1014
vhpiBitStringLiteralK = 1015
vhpiBlockConfigK = 1016
vhpiBlockStmtK = 1017
vhpiBranchK = 1018
vhpiCallbackK = 1019
vhpiCaseStmtK = 1020
vhpiCharLiteralK = 1021
vhpiCompConfigK = 1022
vhpiCompDeclK = 1023
vhpiCompInstStmtK = 1024
vhpiCondSigAssignStmtK = 1025
vhpiCondWaveformK = 1026
vhpiConfigDeclK = 1027
vhpiConstDeclK = 1028
vhpiConstParamDeclK = 1029
vhpiConvFuncK = 1030
vhpiDerefObjK = 1031
vhpiDisconnectSpecK = 1032
vhpiDriverK = 1033
vhpiDriverCollectionK = 1034
vhpiElemAssocK = 1035
vhpiElemDeclK = 1036
vhpiEntityClassEntryK = 1037
vhpiEntityDeclK = 1038
vhpiEnumLiteralK = 1039
vhpiEnumRangeK = 1040
vhpiEnumTypeDeclK = 1041
vhpiExitStmtK = 1042
vhpiFileDeclK = 1043
vhpiFileParamDeclK = 1044
vhpiFileTypeDeclK = 1045
vhpiFloatRangeK = 1046
vhpi_FloatTypeDeclK = 1047
vhpi_ForGenerateK = 1048
vhpi_ForLoopK = 1049
vhpi_ForeignfK = 1050
vhpi_FuncCallK = 1051
vhpi_FuncDeclK = 1052
vhpi_GenericDeclK = 1053
vhpi_GroupDeclK = 1054
vhpi_GroupTempDeclK = 1055
vhpi_IfGenerateK = 1056
vhpi_IfStmtK = 1057
vhpi_InPortK = 1058
vhpi_IndexedNameK = 1059
vhpi_IntLiteralK = 1060
vhpi_IntRangeK = 1061
vhpi_IntTypeDeclK = 1062
vhpi_IteratorK = 1063
vhpi_LibraryDeclK = 1064
vhpi_LoopStmtK = 1065
vhpi_NextStmtK = 1066
vhpi_NullLiteralK = 1067
vhpi_NullStmtK = 1068
vhpi_OperatorK = 1069
vhpi_OthersK = 1070
vhpi_OutPortK = 1071
vhpi_PackBodyK = 1072
vhpi_PackDeclK = 1073
vhpi_PackInstK = 1074
vhpi_ParamAttrNameK = 1075
vhpi_PhysLiteralK = 1076
vhpi_PhysRangeK = 1077
vhpi_PhysTypeDeclK = 1078
vhpi_PortDeclK = 1079
vhpi_ProcCallStmtK = 1080
vhpi_ProcDeclK = 1081
vhpi_ProcessStmtK = 1082
vhpiProtectedTypeK = 1083
vhpiProtectedTypeBodyK = 1084
vhpiProtectedTypeDeclK = 1085
vhpiRealLiteralK = 1086
vhpiRecordTypeDeclK = 1087
vhpiReportStmtK = 1088
vhpiReturnStmtK = 1089
vhpiRootInstK = 1090
vhpiSelectSigAssignStmtK = 1091
vhpiSelectWaveformK = 1092
vhpiSelectedNameK = 1093
vhpiSigDeclK = 1094
vhpiSigParamDeclK = 1095
vhpiSimpAttrNameK = 1096
vhpiSimpleSigAssignStmtK = 1097
vhpiSliceNameK = 1098
vhpiStringLiteralK = 1099
vhpiSubpBodyK = 1100
vhpiSubtypeDeclK = 1101
vhpiSubtypeIndicK = 1102
vhpiToolK = 1103
vhpiTransactionK = 1104
vhpiTypeConvK = 1105
vhpiUnaryExprK = 1106
vhpiUnitDeclK = 1107
vhpiUserAttrNameK = 1108
vhpiVarAssignStmtK = 1109
vhpiVarDeclK = 1110
vhpiVarParamDeclK = 1111
vhpiWaitStmtK = 1112
vhpiWaveformElemK = 1113
vhpiWhileLoopK = 1114
vhpiQualifiedExprK = 1115
vhpiUseClauseK = 1116
vhpiVerilog = 1117
vhpiEdifUnit = 1118
vhpiCollectionK = 1119
vhpiVHDL = 1120
vhpiSystemC = 1121

enum vhpiOneToOneT
    Values:
    vhpiAbstractLiteral = 1301
    vhpiActual = 1302
    vhpiAll = 1303
    vhpiAttrDecl = 1304
    vhpiAttrSpec = 1305
    vhpiBaseType = 1306
    vhpiBaseUnit = 1307
    vhpiBasicSignal = 1308
    vhpiBlockConfig = 1309
    vhpiCaseExpr = 1310
    vhpiCondExpr = 1311
    vhpiConfigDecl = 1312
    vhpiConfigSpec = 1313
    vhpiConstraint = 1314
    vhpiContributor = 1315
    vhpiCurCallback = 1316
    vhpiCurEqProcess = 1317
    vhpiCurStackFrame = 1318
    vhpiDerefObj = 1319
    vhpiDecl = 1320
    vhpiDesignUnit = 1321
    vhpiDownStack = 1322
    vhpiElemSubtype = 1323
    vhpiEntityAspect = 1324
    vhpiEntityDecl = 1325
    vhpiEqProcessStmt = 1326
    vhpiExpr = 1327
    vhpiFormal = 1328
    vhpiFuncDecl = 1329
    vhpiGroupTempDecl = 1330
    vhpiGuardExpr = 1331
<table>
<thead>
<tr>
<th>vhpiGuardSig = 1332</th>
</tr>
</thead>
<tbody>
<tr>
<td>vhpiImmRegion = 1333</td>
</tr>
<tr>
<td>vhpiInPort = 1334</td>
</tr>
<tr>
<td>vhpiInitExpr = 1335</td>
</tr>
<tr>
<td>vhpiIterScheme = 1336</td>
</tr>
<tr>
<td>vhpiLeftExpr = 1337</td>
</tr>
<tr>
<td>vhpiLexicalScope = 1338</td>
</tr>
<tr>
<td>vhpiLhsExpr = 1339</td>
</tr>
<tr>
<td>vhpiLocal = 1340</td>
</tr>
<tr>
<td>vhpiLogicalExpr = 1341</td>
</tr>
<tr>
<td>vhpiName = 1342</td>
</tr>
<tr>
<td>vhpiOperator = 1343</td>
</tr>
<tr>
<td>vhpiOthers = 1344</td>
</tr>
<tr>
<td>vhpiOutPort = 1345</td>
</tr>
<tr>
<td>vhpiParamDecl = 1346</td>
</tr>
<tr>
<td>vhpiParamExpr = 1347</td>
</tr>
<tr>
<td>vhpiParent = 1348</td>
</tr>
<tr>
<td>vhpiPhysLiteral = 1349</td>
</tr>
<tr>
<td>vhpiPrefix = 1350</td>
</tr>
<tr>
<td>vhpiPrimaryUnit = 1351</td>
</tr>
<tr>
<td>vhpiProtectedTypeBody = 1352</td>
</tr>
<tr>
<td>vhpiProtectedTypeDecl = 1353</td>
</tr>
<tr>
<td>vhpiRejectTime = 1354</td>
</tr>
<tr>
<td>vhpiReportExpr = 1355</td>
</tr>
<tr>
<td>vhpiResolFunc = 1356</td>
</tr>
<tr>
<td>vhpiReturnExpr = 1357</td>
</tr>
<tr>
<td>vhpiReturnTypeMark = 1358</td>
</tr>
<tr>
<td>vhpiRhsExpr = 1359</td>
</tr>
<tr>
<td>vhpiRightExpr = 1360</td>
</tr>
<tr>
<td>vhpiRootInst = 1361</td>
</tr>
<tr>
<td>vhpiSelectExpr = 1362</td>
</tr>
<tr>
<td>vhpiSeverityExpr = 1363</td>
</tr>
<tr>
<td>vhpiSimpleName = 1364</td>
</tr>
<tr>
<td>vhpiSubpBody = 1365</td>
</tr>
<tr>
<td>vhpiSubpDecl = 1366</td>
</tr>
<tr>
<td>vhpiSubtype = 1367</td>
</tr>
</tbody>
</table>
vhpiSuffix = 1368
vhpiTimeExpr = 1369
vhpiTimeOutExpr = 1370
vhpiTool = 1371
vhpiType = 1372
vhpiTypeMark = 1373
vhpiTypespec
vhpiUnitDecl = 1374
vhpiUpStack = 1375
vhpiUpperRegion = 1376
vhpiUse = 1377
vhpiValExpr = 1378
vhpiValSubtype = 1379
vhpiElemType = 1380
vhpiFirstNamedType = 1381
vhpiReturnType = 1382
vhpiValType = 1383
vhpiCurRegion = 1384

enum vhpiOneToManyT

Values:
vhpiAliasDecls = 1501
vhpiArgvs = 1502
vhpiAttrDecls = 1503
vhpiAttrSpecs = 1504
vhpiBasicSignals = 1505
vhpiBlockStmts = 1506
vhpiBranchs = 1507
vhpiChoices = 1509
vhpiCompInstStmts = 1510
vhpiCondExprs = 1511
vhpiCondWaveforms = 1512
vhpiConfigItems = 1513
vhpiConfigSpecs = 1514
vhpiConstDecls = 1515
vhpiConstraints = 1516
vhpiContributors = 1517
vhpiDecls = 1519
vhpiDepUnits = 1520
vhpiDesignUnits = 1521
vhpiDrivenSigs = 1522
vhpiDrivers = 1523
vhpiElemAssocs = 1524
vhpiEntityClassEntries = 1525
vhpiEntityDesignators = 1526
vhpiEnumLiterals = 1527
vhpiForeignfs = 1528
vhpiGenericAssocs = 1529
vhpiGenericDecls = 1530
vhpiIndexExprs = 1531
vhpiIndexedNames = 1532
vhpiInternalRegions = 1533
vhpiMembers = 1534
vhpiPackInsts = 1535
vhpiParamAssocs = 1536
vhpiParamDecls = 1537
vhpiPortAssocs = 1538
vhpiPortDecls = 1539
vhpiRecordElems = 1540
vhpiSelectWaveforms = 1541
vhpiSelectedNames = 1542
vhpiSensitivitys = 1543
vhpiSeqStmts = 1544
vhpiSigAttrs = 1545
vhpiSigDecls = 1546
vhpiSigNames = 1547
vhpiSignals = 1548
vhpiSpecNames = 1549
vhpiSpecs = 1550
vhpiStmts = 1551
vhpiTransactions = 1552
vhpiTypeMarks = 1553
vhpiUnitDecls = 1554
enum vhpiIntPropertyT
Values:
vhpiAccessP = 1001
vhpiArgcP = 1002
vhpiAttrKindP = 1003
vhpiBaseIndexP = 1004
vhpiBeginLineNoP = 1005
vhpiEndLineNoP = 1006
vhpiEntityClassP = 1007
vhpiForeignKindP = 1008
vhpiFrameLevelP = 1009
vhpiGenerateIndexP = 1010
vhpiIntValP = 1011
vhpiIsAnonymousP = 1012
vhpiIsBasicP = 1013
vhpiIsCompositeP = 1014
vhpiIsDefaultP = 1015
vhpiIsDeferredP = 1016
vhpiIsDiscreteP = 1017
vhpiIsForcedP = 1018
vhpiIsForeignP = 1019
vhpiIsGuardedP = 1020
vhpiIsImplicitDeclP = 1021
vhpiIsInvalidP = 1022
vhpiIsLocalP = 1023
vhpiIsNamedP = 1024
vhpiIsNullP = 1025
vhpiIsOpenP = 1026
vhpiIsPLIP = 1027
vhpiIsPassiveP = 1028
vhpiIsPostponedP = 1029
vhpiIsProtectedTypeP = 1030
vhpiIsPureP = 1031
vhpiIsResolvedP = 1032
vhpiIsScalarP = 1033
vhpiIsSeqStmtP = 1034
vhpiIsSharedP = 1035
vhpiIsTransportP = 1036
vhpiIsUnaffectedP = 1037
vhpiIsUnconstrainedP = 1038
vhpiIsUninstantiatedP = 1039
vhpiIsUpP = 1040
vhpiIsVitalP = 1041
vhpiIteratorTypeP = 1042
vhpiKindP = 1043
vhpiLeftBoundP = 1044
vhpiLevelP = 1045
vhpiLineNoP = 1046
vhpiLineOffsetP = 1047
vhpiLoopIndexP = 1048
vhpiModeP = 1049
vhpiNumDimensionsP = 1050
vhpiNumFieldsP = 1051
vhpiNumGensP = 1052
vhpiNumLiteralsP = 1053
vhpiNumMembersP = 1054
vhpiNumParamsP = 1055
vhpiNumPortsP = 1056
vhpiOpenModeP = 1057
vhpiPhaseP = 1058
vhpiPositionP = 1059
vhpiPredefAttrP = 1060
vhpiReasonP = 1062
vhpiRightBoundP = 1063
vhpiSigKindP = 1064
vhpiSizeP = 1065
vhpiStartLineNoP = 1066
vhpiStateP = 1067
vhpiStaticnessP = 1068
vhpiVHDVersionP = 1069
vhpiIdP = 1070
vhpiCapabilitiesP = 1071
vhpiIsStdLogicP = 1072
vhpiIsStdULogicP = 1073
vhpiIsStdLogicVectorP = 1074
vhpiIsStdULogicVectorP = 1075
vhpiLanguageP = 1200

enum vhpiStrPropertyT
Values:
vhpiCaseNameP = 1301
vhpiCompNameP = 1302
vhpiDefNameP = 1303
vhpiFileNameP = 1304
vhpiFullCaseNameP = 1305
vhpiFullNameP = 1306
vhpiKindStrP = 1307
vhpiLabelNameP = 1308
vhpiLibLogicalNameP = 1309
vhpiLibPhysicalNameP = 1310
vhpiLogicalNameP = 1311
vhpiLoopLabelNameP = 1312
vhpiNameP = 1313
vhpiOpNameP = 1314
vhpiStrValP = 1315
vhpiToolVersionP = 1316
vhpiUnitNameP = 1317
vhpiSaveRestartLocationP = 1318
vhpiFullVlogNameP = 1500
vhpiFullVHDLNameP = 1501
vhpiFullLSNameP = 1502
vhpiFullLSCaseNameP = 1503

enum vhpiRealPropertyT
  Values:
  vhpiFloatLeftBoundP = 1601
  vhpiFloatRightBoundP = 1602
  vhpiRealValP = 1603

enum vhpiPhysPropertyT
  Values:
  vhpiPhysLeftBoundP = 1651
  vhpiPhysPositionP = 1652
  vhpiPhysRightBoundP = 1653
  vhpiPhysValP = 1654
  vhpiPrecisionP = 1655
  vhpiSimTimeUnitP = 1656
  vhpiResolutionLimitP = 1657

enum vhpiCapabilityT
  Values:
  vhpiProvidesHierarchy = 1
  vhpiProvidesStaticAccess = 2
  vhpiProvidesConnectivity = 4
  vhpiProvidesPostAnalysis = 8
  vhpiProvidesForeignModel = 16
  vhpiProvidesAdvancedForeignModel = 32
  vhpiProvidesSaveRestart = 64
  vhpiProvidesReset = 128
  vhpiProvidesDebugRuntime = 256
  vhpiProvidesAdvancedDebugRuntime = 512
  vhpiProvidesDynamicElab = 1024

enum vhpiOpenModeT
  Values:
  vhpiInOpen = 1001
  vhpiOutOpen = 1002
  vhpiReadOpen = 1003
  vhpiWriteOpen = 1004
  vhpiAppendOpen = 1005
enum vhpiModeT
    Values:
    vhpiInMode = 1001
    vhpiOutMode = 1002
    vhpiInoutMode = 1003
    vhpiBufferMode = 1004
    vhpiLinkageMode = 1005

enum vhpiSigKindT
    Values:
    vhpiRegister = 1001
    vhpiBus = 1002
    vhpiNormal = 1003

enum vhpiStaticnessT
    Values:
    vhpiLocallyStatic = 1001
    vhpiGloballyStatic = 1002
    vhpiDynamic = 1003

enum vhpiPredefAttrT
    Values:
    vhpiActivePA = 1001
    vhpiAscendingPA = 1002
    vhpiBasePA = 1003
    vhpiDelayedPA = 1004
    vhpiDrivingPA = 1005
    vhpiDriving_valuePA = 1006
    vhpiEventPA = 1007
    vhpiHighPA = 1008
    vhpiImagePA = 1009
    vhpiInstance_namePA = 1010
    vhpiLast_activePA = 1011
    vhpiLast_eventPA = 1012
    vhpiLast_valuePA = 1013
    vhpiLeftPA = 1014
    vhpiLeftofPA = 1015
    vhpiLengthPA = 1016
    vhpiLowPA = 1017
    vhpiPath_namePA = 1018
vhpiPosPA = 1019
vhpiPredPA = 1020
vhpiQuietPA = 1021
vhpiRangePA = 1022
vhpiReverse_rangePA = 1023
vhpiRightPA = 1024
vhpiRightofPA = 1025
vhpiSimple_namePA = 1026
vhpiStablePA = 1027
vhpiSuccPA = 1028
vhpiTransactionPA = 1029
vhpiValPA = 1030
vhpiValuePA = 1031

enum vhpiAttrKindT
    Values:
    vhpiFunctionAK = 1
    vhpiRangeAK = 2
    vhpiSignalAK = 3
    vhpiTypeAK = 4
    vhpiValueAK = 5

enum vhpiEntityClassT
    Values:
    vhpiEntityEC = 1001
    vhpiArchitectureEC = 1002
    vhpiConfigurationEC = 1003
    vhpiProcedureEC = 1004
    vhpiFunctionEC = 1005
    vhpiPackageEC = 1006
    vhpiTypeEC = 1007
    vhpiSubtypeEC = 1008
    vhpiConstantEC = 1009
    vhpiSignalEC = 1010
    vhpiVariableEC = 1011
    vhpiComponentEC = 1012
    vhpiLabelEC = 1013
    vhpiLiteralEC = 1014
    vhpiUnitsEC = 1015
vhpiFileEC = 1016
vhpiGroupEC = 1017

enum vhpiAccessT

Values:
vhpiRead = 1
vhpiWrite = 2
vhpiConnectivity = 4
vhpiNoAccess = 8

enum vhpiStateT

Values:
vhpiEnable
vhpiDisable
vhpiMature

enum vhpiCompInstKindT

Values:
vhpiDirect
vhpiComp
vhpiConfig

enum vhpiPhaseT

Values:
vhpiRegistrationPhase = 1
vhpiAnalysisPhase = 2
vhpiElaborationPhase = 3
vhpiInitializationPhase = 4
vhpiSimulationPhase = 5
vhpiTerminationPhase = 6
vhpiSavePhase = 7
vhpiRestartPhase = 8
vhpiResetPhase = 9

enum vhpiSeverityT

Values:
vhpiNote = 1
vhpiWarning = 2
vhpiError = 3
vhpiFailure = 6
vhpiSystem = 4
vhpiInternal = 5
enum vhpiAutomaticRestoreT
  
  Values:
  
  vhpiRestoreAll = 1
  vhpiRestoreUserData = 2
  vhpiRestoreHandles = 4
  vhpiRestoreCallbacks = 8

enum vhpiPutValueModeT
  
  Values:
  
  vhpiDeposit
  vhpiDepositPropagate
  vhpiForce
  vhpiForcePropagate
  vhpiRelease
  vhpiSizeConstraint

enum vhpiDelayModeT
  
  Values:
  
  vhpiInertial
  vhpiTransport

enum vhpiSimControlT
  
  Values:
  
  vhpiStop = 0
  vhpiFinish = 1
  vhpiReset = 2

enum vhpiForeignT
  
  Values:
  
  vhpiArchF = 1
  vhpiArchFK = 1
  vhpiFuncF = 2
  vhpiFuncFK = 2
  vhpiProcF = 3
  vhpiProcFK = 3
  vhpiLibF = 4
  vhpiAppF = 5
Functions

```c
EXTERN int vhpi_assert(vhpiSeverityT severity, const char * formatmsg, ...)
EXTERN vhpiHandleT vhpi_register_cb(vhpiCbDataT * cb_data_p, int32_t flags)
EXTERN int vhpi_remove_cb(vhpiHandleT cb_obj)
EXTERN int vhpi_disable_cb(vhpiHandleT cb_obj)
EXTERN int vhpi_enable_cb(vhpiHandleT cb_obj)
EXTERN int vhpi_get_cb_info(vhpiHandleT object, vhpiCbDataT * cb_data_p)
EXTERN int vhpi_sens_first(vhpiValueT * sens)
EXTERN int vhpi_sens_zero(vhpiValueT * sens)
EXTERN int vhpi_sens_clr(int obj, vhpiValueT * sens)
EXTERN int vhpi_sens_set(int obj, vhpiValueT * sens)
EXTERN int vhpi_sens_isset(int obj, vhpiValueT * sens)
EXTERN vhpiHandleT vhpi_handle_by_name(const char * name, vhpiHandleT scope)
EXTERN vhpiHandleT vhpi_handle_by_index(vhpiOneToManyT itRel, vhpiHandleT parent, int32_t indx)
EXTERN vhpiHandleT vhpi_handle(vhpiOneToOneT type, vhpiHandleT referenceHandle)
EXTERN vhpiHandleT vhpi_iterator(vhpiOneToManyT type, vhpiHandleT referenceHandle)
EXTERN vhpiHandleT vhpi_scan(vhpiHandleT iterator)
EXTERN vhpiIntT vhpi_get(vhpiIntPropertyT property, vhpiHandleT object)
EXTERN const vhpiCharT* vhpi_get_str(vhpiStrPropertyT property, vhpiHandleT object)
EXTERN vhpiRealT vhpi_get_real(vhpiRealPropertyT property, vhpiHandleT object)
EXTERN vhpiPhysT vhpi_get_phys(vhpiPhysPropertyT property, vhpiHandleT object)
EXTERN int vhpi_protected_call(vhpiHandleT varHdl, vhpiUserFctT userFct, void * userData)
EXTERN int vhpi_get_value(vhpiHandleT expr, vhpiValueT * value_p)
EXTERN int vhpi_put_value(vhpiHandleT object, vhpiValueT * value_p, vhpiPutValueModeT flags)
EXTERN int vhpi_schedule_transaction(vhpiHandleT drivHdl, vhpiValueT * value_p, uint32_t numValues, vhpiTimeT * delayp, vhpiDelayModeT delayMode, vhpiTimeT * pulseRejp)
EXTERN int vhpi_format_value(const vhpiValueT * in_value_p, vhpiValueT * out_value_p)
EXTERN void vhpi_get_time(vhpiTimeT * time_p, long * cycles)
EXTERN int vhpi_get_next_time(vhpiTimeT * time_p)
EXTERN int vhpi_control(vhpiSimControlT command, ...) 
EXTERN int vhpi_sim_control(vhpiSimControlT command)
EXTERN int vhpi_printf(const char * format, ...)
EXTERN int vhpi_vprintf(const char * format, va_list args)
EXTERN int vhpi_is_printable(char ch)
EXTERN int vhpi_compare_handles(vhpiHandleT handle1, vhpiHandleT handle2)
EXTERN int vhpi_check_error(vhpiErrorInfoT * error_info_p)
EXTERN int vhpi_release_handle(vhpiHandleT object)
```
XXTERN vhpiHandleT vhpi_create(vhpiClassKindT kind, vhpiHandleT handle1, vhpiHandleT handle2)
XXTERN vhpiHandleT vhpi_register_foreignf(vhpiForeignDataT * foreignDatap)
XXTERN int vhpi_get_foreignf_info(vhpiHandleT hdl, vhpiForeignDataT * foreignDatap)
XXTERN int vhpi_get_foreign_info(vhpiHandleT hdl, vhpiForeignDataT * foreignDatap)
XXTERN size_t vhpi_get_data(int32_t id, void * dataLoc, size_t numBytes)
XXTERN size_t vhpi_put_data(int32_t id, void * dataLoc, size_t numBytes)
XXTERN vhpiHandleT vhpi_get_cause_instance(vhpiHandleT sigHandle)
XXTERN int vhpi_get_cause(vhpiHandleT sigHandle, unsigned int ** p2MagicNumbersBuffer)
XXTERN int vhpi_get_cause_info(const unsigned int ** pn2MagicNumbers, int nBufLen, char * pszHierScopeBuf, int nFilePathBufLen, char * pszSourceFilePathBuf, int * pnLineNr)
XXTERN vhpiIntT vhpi_value_size(vhpiHandleT objHdl, vhpiFormatT format)

Variables

PLI_VEXTERN PLI_DLLISPEC const vhpiPhysT vhpiFS
PLI_VEXTERN PLI_DLLISPEC const vhpiPhysT vhpiPS
PLI_VEXTERN PLI_DLLISPEC const vhpiPhysT vhpiNS
PLI_VEXTERN PLI_DLLISPEC const vhpiPhysT vhpiUS
PLI_VEXTERN PLI_DLLISPEC const vhpiPhysT vhpiMS
PLI_VEXTERN PLI_DLLISPEC const vhpiPhysT vhpiS
PLI_VEXTERN PLI_DLLISPEC const vhpiPhysT vhpiMN
PLI_VEXTERN PLI_DLLISPEC const vhpiPhysT vhpiHR

struct vhpiPhysS

Public Members

int32_t high
uint32_t low

struct vhpiTimeS

Public Members

uint32_t high
uint32_t low

struct vhpiValueS

8.1. API Documentation
Public Members

vhpiFormatT format
size_t bufSize
int32_t numElems
vhpiPhysT unit
vhpiEnumT_enumv
vhpiEnumT_enumvs
vhpiSmallEnumT_smallenumv
vhpiSmallEnumT_smallenumvs
vhpiIntT intg
vhpiIntT_intgs
vhpiLongIntT_longintg
vhpiLongIntT_longintgs
vhpiRealT real
vhpiRealT_reals
vhpiSmallPhysT_smallphys
vhpiSmallPhysT_smallphys
vhpiPhysT phys
vhpiPhysT_physs
vhpiTimeT time
vhpiTimeT_times
vhpiCharT ch
vhpiCharT_str
void *ptr
void **ptrs
union vhpiValueS::[anonymous] value

struct vhpiErrorInfoS

Public Members

vhpiSeverityT severity
char *message
char *str
char *file
int32_t line

struct vhpiCbDataS

142 Chapter 8. C Code Library Reference
Public Members

int32_t reason
void (*cb_rtn)(const struct vhpiCbDataS *)
vhpiHandleT obj
vhpiTimeT *time
vhpiValueT *value
void *user_data
struct vhpiForeignDataS

Public Members

vhpiForeignT kind
char *libraryName
char *modelName
void (*elabf)(const struct vhpiCbDataS *cb_data_p)
void (*execf)(const struct vhpiCbDataS *cb_data_p)

File vpi_user.h

Defines

SVPI_TYPES
PLI_TYPES
PLI_DLLISPEC
PLI_DLLESPEC
PLI_EXTERN
PLI_VEXTERN extern
PLI_PROTOTYPES
PROTO_PARAMS (params) params
XXTERN PLI_EXTERN PLI_DLLISPEC
EETERN PLI_EXTERN PLI_DLLESPEC
vpiAlways 1 /* always construct */
vpiAssignStmt 2 /* quasi-continuous assignment */
vpiAssignment 3 /* procedural assignment */
vpiBegin 4 /* block statement */
vpiCase 5 /* case statement */
vpiCaseItem 6 /* case statement item */
vpiConstant 7 /* numerical constant or literal string */

8.1. API Documentation
vpiContAssign 8 /* continuous assignment */
vpiDeassign 9 /* deassignment statement */
vpiDefParam 10 /* defparam */
vpiDelayControl 11 /* delay statement (e.g. #10) */
vpiDisable 12 /* named block disable statement */
vpiEventControl 13 /* wait on event, e.g. @e */
vpiEventStmt 14 /* event trigger, e.g. ->e */
vpiFor 15 /* for statement */
vpiForce 16 /* force statement */
vpiForever 17 /* forever statement */
vpiFork 18 /* fork-join block */
vpiFuncCall 19 /* HDL function call */
vpiFunction 20 /* HDL function */
vpiGate 21 /* primitive gate */
vpiIf 22 /* if statement */
vpiIfElse 23 /* if-else statement */
vpiInitial 24 /* initial construct */
vpiIntegerVar 25 /* integer variable */
vpiInterModPath 26 /* intermodule wire delay */
vpiIterator 27 /* iterator */
vpiIODecl 28 /* input/output declaration */
vpiMemory 29 /* behavioral memory */
vpiMemoryWord 30 /* single word of memory */
vpiModPath 31 /* module path for path delays */
vpiModule 32 /* module instance */
vpiNamedBegin 33 /* named block statement */
vpiNamedEvent 34 /* event variable */
vpiNamedFork 35 /* named fork-join block */
vpiNet 36 /* scalar or vector net */
vpiNetBit 37 /* bit of vector net */
vpiNullStmt 38 /* a semicolon. Ie. #10 ; */
vpiOperation 39 /* behavioral operation */
vpiParamAssign 40 /* module parameter assignment */
vpiParameter 41 /* module parameter */
vpiPartSelect 42 /* part-select */
vpiPathTerm 43 /* terminal of module path */
vpiPort 44 /* module port */
vpiPortBit 45 /* bit of vector module port */
vpiPrimTerm 46 /* primitive terminal */
vpiRealVar 47 /* real variable */
vpiReg 48 /* scalar or vector reg */
vpiRegBit 49 /* bit of vector reg */
vpiRelease 50 /* release statement */
vpiRepeat 51 /* repeat statement */
vpiRepeatControl 52 /* repeat control in an assign stmt */
vpiSchedEvent 53 /* vpi_put_value() event */
vpiSpecParam 54 /* specparam */
vpiSwitch 55 /* transistor switch */
vpiSysFuncCall 56 /* system function call */
vpiSysTaskCall 57 /* system task call */
vpiTableEntry 58 /* UDP state table entry */
vpiTask 59 /* HDL task */
vpiTaskCall 60 /* HDL task call */
vpiTchk 61 /* timing check */
vpiTchkTerm 62 /* terminal of timing check */
vpiTimeVar 63 /* time variable */
vpiTimeQueue 64 /* simulation event queue */
vpiUdp 65 /* user-defined primitive */
vpiUdpDefn 66 /* UDP definition */
vpiUserSystf 67 /* user defined system task or function */
vpiVarSelect 68 /* variable array selection */
vpiWait 69 /* wait statement */
vpiWhile 70 /* while statement */
vpiAttribute 105 /* attribute of an object */
vpiBitSelect 106 /* Bit-select of parameter, var select */
vpiCallback 107 /* callback object */
vpiDelayTerm 108 /* Delay term which is a load or driver */
vpiDelayDevice 109 /* Delay object within a net */
vpiFrame 110 /* reentrant task/func frame */
vpiGateArray 111 /* gate instance array */
vpiModuleArray 112 /* module instance array */
vpiPrimitiveArray 113 /* vpiprimitiveArray type */

8.1. API Documentation
vpiNetArray /* multidimensional net */
vpiRange /* range declaration */
vpiRegArray /* multidimensional reg */
vpiSwitchArray /* switch instance array */
vpiUdpArray /* UDP instance array */
vpiContAssignBit /* Bit of a vector continuous assignment */
vpiNamedEventArray /* multidimensional named event */
vpiIndexedPartSelect /* Indexed part-select object */
vpiGenScopeArray /* array of generated scopes */
vpiGenScope /* A generated scope */
vpiGenVar /* Object used to instantiate gen scopes */
vpiCondition /* condition expression */
vpiDelay /* net or gate delay */
vpiElseStmt /* else statement */
vpiForIncStmt /* increment statement in for loop */
vpiForInitStmt /* initialization statement in for loop */
vpiHighConn /* higher connection to port */
vpiLhs /* left-hand side of assignment */
vpiIndex /* index of var select, bit-select, etc. */
vpiLeftRange /* left range of vector or part-select */
vpiLowConn /* lower connection to port */
vpiParent /* parent object */
vpiRhs /* right-hand side of assignment */
vpiRightRange /* right range of vector or part-select */
vpiScope /* containing scope object */
vpiSysTfCall /* task function call */
vpiTchkDataTerm /* timing check data term */
vpiTchkNotifier /* timing check notifier */
vpiTchkRefTerm /* timing check reference term */
vpiArgument /* argument to (system) task/function */
vpiBit /* bit of vector net or port */
vpiDriver /* driver for a net */
vpiInternalScope /* internal scope in module */
vpiLoad /* load on net or reg */
vpiModDataPathIn /* data terminal of a module path */
vpiModPathIn /* Input terminal of a module path */
vpiModPathOut  96 /* output terminal of a module path */
vpiOperand  97 /* operand of expression */
vpiPortInst  98 /* connected port instance */
vpiProcess  99 /* process in module */
vpiVariables  100 /* variables in module */
vpiUse  101 /* usage */
vpiExpr  102 /* connected expression */
vpiPrimitive  103 /* primitive (gate, switch, UDP) */
vpiStmt  104 /* statement in process or task */
vpiActiveTimeFormat  119 /* active $timeformat() system task */
vpiInTerm  120 /* To get to a delay device’s drivers. */
vpiInstanceArray  121 /* vpiInstance arrays */
vpiLocalDriver  122 /* local drivers (within a module */
vpiLocalLoad  123 /* local loads (within a module */
vpiOutTerm  124 /* To get to a delay device’s loads. */
vpiPorts  125 /* Module port */
vpiSimNet  126 /* simulated net after collapsing */
vpiTaskFunc  127 /* HDL task or function */
vpiBaseExpr  131 /* Indexed part-select’s base expression */
vpiWidthExpr  132 /* Indexed part-select’s width expression */
vpiAutomatics  136 /* Automatic variables of a frame */
vpiUndefined  -1 /* undefined property */
vpiType  1 /* type of object */
vpiName  2 /* local name of object */
vpiFullName  3 /* full hierarchical name */
vpiSize  4 /* size of gate, net, port, etc. */
vpiFile  5 /* File name in which the object is used*/
vpiLineNo  6 /* line number where the object is used */
vpiTopModule  7 /* top-level module (boolean) */
vpiCellInstance  8 /* cell (boolean) */
vpiDefName  9 /* module definition name */
vpiProtected  10 /* source protected module (boolean) */
vpiTimeUnit  11 /* module time unit */
vpiTimePrecision  12 /* module time precision */
vpiDefNetType  13 /* default net type */
vpiUnconnDrive  14 /* unconnected port drive strength */
vpiHighZ 1 /* No default drive given */
vpiPull1 2 /* default pull1 drive */
vpiPull0 3 /* default pull0 drive */
vpiDefFile 15 /* File name where the module is defined */
vpiDefLineNo 16 /* line number for module definition */
vpiDefDelayMode 47 /* Default delay mode for a module */
vpiDelayModeNone 1 /* no delay mode specified */
vpiDelayModePath 2 /* path delay mode */
vpiDelayModeDistrib 3 /* distributed delay mode */
vpiDelayModeUnit 4 /* unit delay mode */
vpiDelayModeZero 5 /* zero delay mode */
vpiDelayModeMTM 6 /* min:typ:max delay mode */
vpiDefDecayTime 48 /* Default decay time for a module */
vpiScalar 17 /* scalar (boolean) */
vpiVector 18 /* vector (boolean) */
vpiExplicitName 19 /* port is explicitly named */
vpiDirection 20 /* direction of port: */
vpiInput 1 /* input */
vpiOutput 2 /* output */
vpiInout 3 /* inout */
vpiMixedIO 4 /* mixed input-output */
vpiNoDirection 5 /* no direction */
vpiConnByName 21 /* connected by name (boolean) */
vpiNetType 22 /* net subtypes: */
vpiWire 1 /* wire net */
vpiWand 2 /* wire-and net */
vpiWor 3 /* wire-or net */
vpiTri 4 /* three-state net */
vpiTri0 5 /* pull-down net */
vpiTri1 6 /* pull-up net */
vpiTriReg 7 /* tri state reg net */
vpiTriAnd 8 /* three-state wire-and net */
vpiTriOr 9 /* three-state wire-or net */
vpiSupply1 10 /* supply 1 net */
vpiSupply0 11 /* supply zero net */
vpiNone 12 /* no default net type (1364-2001) */
vpiUwire 13 /* unresolved wire net (1364-2005) */
vpiExplicitScalar 23 /* explicitly scalar (boolean) */
vpiExplicitVector 24 /* explicitly vectored (boolean) */
vpiExpanded 25 /* expanded vector net (boolean) */
vpiImplicitDecl 26 /* implicitly declared net (boolean) */
vpiChargeStrength 27 /* charge decay strength of net */
vpiArray 28 /* variable array (boolean) */
vpiPortIndex 29 /* Port index */
vpiTermIndex 30 /* Index of a primitive terminal */
vpiStrength0 31 /* 0-strength of net or gate */
vpiStrength1 32 /* 1-strength of net or gate */
vpiPrimType 33 /* primitive subtypes: */
vpiAndPrim 1 /* and gate */
vpiNandPrim 2 /* nand gate */
vpiNorPrim 3 /* nor gate */
vpiOrPrim 4 /* or gate */
vpiXorPrim 5 /* xor gate */
vpiXnorPrim 6 /* xnor gate */
vpiBufPrim 7 /* buffer */
vpiNotPrim 8 /* not gate */
vpiBufif0Prim 9 /* zero-enabled buffer */
vpiBufif1Prim 10 /* one-enabled buffer */
vpiNotif0Prim 11 /* zero-enabled not gate */
vpiNotif1Prim 12 /* one-enabled not gate */
vpiNmosPrim 13 /* nmos switch */
vpiPmosPrim 14 /* pmos switch */
vpiCmosPrim 15 /* cmos switch */
vpiRnmosPrim 16 /* resistive nmos switch */
vpiRpmosPrim 17 /* resistive pmos switch */
vpiRcmosPrim 18 /* resistive cmos switch */
vpiRtranPrim 19 /* resistive bidirectional */
vpiRtranif0Prim 20 /* zero-enable resistive bidirectional */
vpiRtranif1Prim 21 /* one-enable resistive bidirectional */
vpiTranPrim 22 /* bidirectional */
vpiTranif0Prim 23 /* zero-enabled bidirectional */
vpiTranif1Prim 24 /* one-enabled bidirectional */
vpiPullupPrim 25 /* pullup */
vpiPulldownPrim 26 /* pulldown */
vpiSeqPrim 27 /* sequential UDP */
vpiCombPrim 28 /* combinational UDP */
vpiPolarity 34 /* polarity of module path... */
vpiDataPolarity 35 /* ... or data path: */
vpiPositive 1 /* positive */
vpiNegative 2 /* negative */
vpiUnknown 3 /* unknown (unspecified) */
vpiEdge 36 /* edge type of module path: */
vpiNoEdge 0x00 /* no edge */
vpiEdge01 0x01 /* 0 -> 1 */
vpiEdge10 0x02 /* 1 -> 0 */
vpiEdge0x 0x04 /* 0 -> x */
vpiEdgex0 0x08 /* x -> 1 */
vpiEdgex1 0x10 /* 1 -> x */
vpiEdgex0x 0x20 /* x -> 0 */
vpiPosedge (vpiEdgex1 | vpiEdge01 | vpiEdge0x)
vpiNegedge (vpiEdgex0 | vpiEdge10 | vpiEdge1x)
vpiAnyEdge (vpiPosedge | vpiNegedge)
vpiPathType 37 /* path delay connection subtypes: */
vpiPathFull 1 /* (a -> b) */
vpiPathParallel 2 /* (a =/> b) */
vpiTchkType 38 /* timing check subtypes: */
vpiSetup 1 /* $setup */
vpiHold 2 /* $hold */
vpiPeriod 3 /* $period */
vpiWidth 4 /* $width */
vpiSkew 5 /* $skew */
vpiRecovery 6 /* $recovery */
vpiNoChange 7 /* $nochange */
vpiSetupHold 8 /* $setuphold */
vpiFullskew 9 /* $fullskew – added for 1364-2001 */
vpiRecrem 10 /* $recrem – added for 1364-2001 */
vpiRemoval 11 /* $removal – added for 1364-2001 */
vpiTimeskew 12 /* $timeskew – added for 1364-2001 */
vpiOpType 39 /* operation subtypes: */
vpiMinusOp 1 /* unary minus */
vpiPlusOp 2 /* unary plus */
vpiNotOp 3 /* unary not */
vpiBitNegOp 4 /* bitwise negation */
vpiUnaryAndOp 5 /* bitwise reduction and */
vpiUnaryNandOp 6 /* bitwise reduction nand */
vpiUnaryOrOp 7 /* bitwise reduction or */
vpiUnaryNorOp 8 /* bitwise reduction nor */
vpiUnaryXorOp 9 /* bitwise reduction xor */
vpiUnaryXNorOp 10 /* bitwise reduction xnor */
vpiSubOp 11 /* binary subtraction */
vpiDivOp 12 /* binary division */
vpiModOp 13 /* binary modulus */
vpiEqOp 14 /* binary equality */
vpiNeqOp 15 /* binary inequality */
vpciCaseEqOp 16 /* case (x and z) equality */
vpciCaseNeqOp 17 /* case inequality */
vpiGTOp 18 /* binary greater than */
vpiGeOp 19 /* binary greater than or equal */
vpiLTOp 20 /* binary less than */
vpiLeOp 21 /* binary less than or equal */
vpiLShiftOp 22 /* binary left shift */
vpiRShiftOp 23 /* binary right shift */
vpiAddOp 24 /* binary addition */
vpiMultOp 25 /* binary multiplication */
vpiLogAndOp 26 /* binary logical and */
vpiLogOrOp 27 /* binary logical or */
vpiBitAndOp 28 /* binary bitwise and */
vpiBitOrOp 29 /* binary bitwise or */
vpiBitXorOp 30 /* binary bitwise xor */
vpiBitXNorOp 31 /* binary bitwise xnor */
vpiBitXnorOp 32 /* added with 1364-2001 */
vpciConditionOp 32 /* ternary conditional */
vpciConcatOp 33 /* n-ary concatenation */
vpiMultiConcatOp 34 /* repeated concatenation */
vpiEventOrOp 35 /* event or */
vpiNullOp 36 /* null operation */
vpiListOp 37 /* list of expressions */
vpiMinTypMaxOp 38 /* min:typ:max: delay expression */
vpiPosedgeOp 39 /* posedge */
vpiNegedgeOp 40 /* negedge */
vpiArithLShiftOp 41 /* arithmetic left shift (1364-2001) */
vpiArithRShiftOp 42 /* arithmetic right shift (1364-2001) */
vpiPowerOp 43 /* arithmetic power op (1364-2001) */
vpiConstType 40 /* constant subtypes: */
vpiDecConst 1 /* decimal integer */
vpiRealConst 2 /* real */
vpiBinaryConst 3 /* binary integer */
vpiOctConst 4 /* octal integer */
vpiHexConst 5 /* hexadecimal integer */
vpiStringConst 6 /* string literal */
vpiIntConst 7 /* HDL integer constant (1364-2001) */
vpiTimeConst 8 /* time constant */
vpiBlocking 41 /* blocking assignment (boolean) */
vpiCaseType 42 /* case statement subtypes: */
vpiCaseExact 1 /* exact match */
vpiCaseX 2 /* ignore X's */
vpiCaseZ 3 /* ignore Z's */
vpiNetDeclAssign 43 /* assign part of decl (boolean) */
vpiFuncType 44 /* HDL function & system function type */
vpiIntFunc 1 /* returns integer */
vpiRealFunc 2 /* returns real */
vpiTimeFunc 3 /* returns time */
vpiSizedFunc 4 /* returns an arbitrary size */
vpiSizedSignedFunc 5 /* returns sized signed value */
vpiSysFuncType vpiFuncType
    alias 1364-1995 system function subtypes to 1364-2001 function subtypes
vpiSysFuncInt vpiIntFunc
vpiSysFuncReal vpiRealFunc
vpiSysFuncTime vpiTimeFunc
vpiSysFuncSized vpiSizedFunc
vpiUserDefn 45 /* user defined system task/func(boolean) */
vpiScheduled 46 /* object still scheduled (boolean) */
vpiActive 49 /* reentrant task/func frame is active */
vpiAutomatic 50 /* task/func obj is automatic */
vpiCell 51 /* configuration cell */
vpiConfig 52 /* configuration config file */
                  indices are constant expressions */

vpiDecompile 54 /* decompile the object */
vpiDefAttribute 55 /* Attribute defined for the obj */
vpiDelayType 56 /* delay subtype */
vpiModPathDelay 1 /* module path delay */
vpiInterModPathDelay 2 /* intermodule path delay */
vpiMIPDelay 3 /* module input port delay */
vpiIteratorType 57 /* object type of an iterator */
vpiLibrary 58 /* configuration library */
vpiMultiArray 59 /* Object is a multidimensional array */
vpiOffset 60 /* offset from LSB */
                  same subtypes as vpiNetType */

vpiSaveRestartID 62 /* unique ID for save/restart data */
vpiSaveRestartLocation 63 /* name of save/restart data file */
                  variable is valid */

vpiValidFalse 0
vpiValidTrue 1
                  the expression class if the object has the signed attribute */
                  localparam */

vpiModPathHasIfNone 71 /* Mod path has an ifnone statement */
vpiIndexedPartSelectType 72 /* Indexed part-select type */
vpiPosIndexed 1 /* +: */

vpiNegIndexed 2 /* -: */
vpiIsMemory 73 /* TRUE for a one-dimensional reg array */
vpiIsProtected 74 /* TRUE for protected design information */
vpiStop 66 /* execute simulator’s $stop */
vpiFinish 67 /* execute simulator’s $finish */
vpiReset 68 /* execute simulator’s $reset */
vpiSetInteractiveScope 69 /* set simulator’s interactive scope */

VPI_MCD_STDOUT 0x00000001

8.1. API Documentation
vpiScaledRealTime 1
vpiSimTime 2
vpiSuppressTime 3
VPI_VECVAL
vpiSupplyDrive 0x80
vpiStrongDrive 0x40
vpiPullDrive 0x20
vpiWeakDrive 0x08
vpiLargeCharge 0x10
vpiMediumCharge 0x04
vpiSmallCharge 0x02
vpiHiZ 0x01
vpiBinStrVal 1
vpiOctStrVal 2
vpiDecStrVal 3
vpiHexStrVal 4
vpiScalarVal 5
vpiIntVal 6
vpiRealVal 7
vpiStringVal 8
vpiVectorVal 9
vpiStrengthVal 10
vpiTimeVal 11
vpiObjTypeVal 12
vpiSuppressVal 13
vpiShortIntVal 14
vpiLongIntVal 15
vpiShortRealVal 16
vpiRawTwoStateVal 17
vpiRawFourStateVal 18
vpiNoDelay 1
vpiInertialDelay 2
vpiTransportDelay 3
vpiPureTransportDelay 4
vpiForceFlag 5
vpiReleaseFlag 6
vpiCancelEvent 7
vpiReturnEvent 0x1000
vpiUserAllocFlag 0x2000
vpiOneValue 0x4000
vpiPropagateOff 0x8000
vpi0 0
vpi1 1
vpiZ 2
vpiX 3
vpiH 4
vpiL 5
vpiDontCare 6
vpiSysTask 1
vpiSysFunc 2
vpiCompile 1
vpiPLI 2
vpiRun 3
vpiNotice 1
vpiWarning 2
vpiError 3
vpiSystem 4
vpiInternal 5
vpiTimePrecision 12 /* module time precision */
cbValueChange 1
cbStmt 2
cbForce 3
cbRelease 4
cbAtStartOfSimTime 5
cbReadWriteSynch 6
cbReadOnlySynch 7
cbNextSimTime 8
cbAfterDelay 9
cbEndOfCompile 10
cbStartOfSimulation 11
cbEndOfSimulation 12
cbError 13
cbTchkViolation 14
cbStartOfSave 15
cbEndOfSave 16
cbStartOfRestart 17
cbEndOfRestart 18
cbStartOfReset 19
cbEndOfReset 20
cbEnterInteractive 21
cbExitInteractive 22
cbInteractiveScopeChange 23
cbUnresolvedSystf 24
cbAssign 25
cbDeassign 26
cbDisable 27
cbPLIError 28
cbSignal 29
cbNBASynch 30
cbAtEndOfSimTime 31

**Typedefs**

typedef int64_t PLI_INT64

typedef uint64_t PLI_UINT64

typedef int PLI_INT32

typedef unsigned int PLI_UINT32

typedef short PLI_INT16

typedef unsigned short PLI_UINT16

typedef char PLI_BYTE8

typedef unsigned char PLI_UBYTE8

typedef PLI_UINT32 *vpiHandle

typedef struct t_vpi_time s_vpi_time

typedef struct t_vpi_time *p_vpi_time

typedef struct t_vpi_delay s_vpi_delay

typedef struct t_vpi_delay *p_vpi_delay

typedef struct t_vpi_vecval s_vpi_vecval

typedef struct t_vpi_vecval *p_vpi_vecval

typedef struct t_vpi_strengthval s_vpi_strengthval
typedef struct t_vpi_strengthval *p_vpi_strengthval
typedef struct t_vpi_value s_vpi_value
typedef struct t_vpi_value *p_vpi_value
typedef struct t_vpi_arrayvalue s_vpi_arrayvalue
typedef struct t_vpi_arrayvalue *p_vpi_arrayvalue
typedef struct t_vpi_systf_data s_vpi_systf_data
typedef struct t_vpi_systf_data *p_vpi_systf_data
typedef struct t_vpi_vlog_info s_vpi_vlog_info
typedef struct t_vpi_vlog_info *p_vpi_vlog_info
typedef struct t_vpi_error_info s_vpi_error_info
typedef struct t_vpi_error_info *p_vpi_error_info
typedef struct t_cb_data s_cb_data
typedef struct t_cb_data *p_cb_data

Functions

XXTERN vpiHandle vpi_register_cb PROTO_PARAMS(( p_cb_data cb_data_p))
XXTERN PLI_INT32 vpi_remove_cb PROTO_PARAMS((vpiHandle cb_obj))
XXTERN void vpi_get_cb_info PROTO_PARAMS((vpiHandle object, p_cb_data cb_data_p))
XXTERN vpiHandle vpi_register_systf PROTO_PARAMS(( p_vpi_systf_data systf_data_p))
XXTERN void vpi_get_systf_info PROTO_PARAMS((vpiHandle object, p_vpi_systf_data systf_data_p))
XXTERN vpiHandle vpi_handle_by_name PROTO_PARAMS((PLI_BYTE8 *name, vpiHandle scope))
XXTERN vpiHandle vpi_handle_by_index PROTO_PARAMS((vpiHandle object, PLI_INT32 indx))
XXTERN vpiHandle vpi_iterate PROTO_PARAMS((PLI_INT32 type, vpiHandle refHandle))
XXTERN vpiHandle vpi_handle_multi PROTO_PARAMS((PLI_INT32 type, vpiHandle refHandle1, vpiHandle refHandle2,...))
XXTERN vpiHandle vpi_scan PROTO_PARAMS((vpiHandle iterator))
XXTERN PLI_BYTE8 *vpi_get_str PROTO_PARAMS((PLI_INT32 property, vpiHandle object))
XXTERN void vpi_put_delays PROTO_PARAMS((vpiHandle object, _p_vpi_delay delay_p))
XXTERN void vpi_get_value PROTO_PARAMS((vpiHandle expr, _p_vpi_value value_p))
XXTERN vpiHandle vpi_put_value PROTO_PARAMS((vpiHandle object, _p_vpi_value value_p, _p_vpi_time time_p, PLI_INT32 flags))
XXTERN void vpi_get_value_array PROTO_PARAMS((vpiHandle expr, _p_vpi_arrayvalue arrayvalue_p, PLI_INT32 *index_p, PLI_UINT32 num))
XXTERN void vpi_put_value_array PROTO_PARAMS((vpiHandle object, _p_vpi_arrayvalue arrayvalue_p, PLI_INT32 *index_p, PLI_UINT32 num))
XXTERN void vpi_get_time PROTO_PARAMS((vpiHandle object, _p_vpi_time time_p))
XXTERN PLI_UINT32 vpi_mcd_open PROTO_PARAMS((const PLI_BYTE8 *fileName))
XXTERN PLI_INT32 vpi_mcd_flush PROTO_PARAMS((PLI_UINT32 mcd))
XXTERN PLI_BYTE8* vpi_mcd_name PROTO_PARAMS((PLI_UINT32 cd))
XXTERN PLI_UINT32 vpi_mcd_printf PROTO_PARAMS((PLI_UINT32 mcd, const PLI_BYTE8 *format,...))
Variables

PLI_VEXTERN PLI_DLLESPEC void(* vlog_startup_routines[])(void)

struct t_vpi_time

Public Members

PLI_INT32 type
PLI_UINT32 high
PLI_UINT32 low
double real

struct t_vpi_delay

Public Members

struct t_vpi_time *da
PLI_INT32 no_of_delays
PLI_INT32 time_type
PLI_INT32 mtm_flag
PLI_INT32 append_flag
PLI_INT32 pulsere_flag

struct t_vpi_vecval
Public Members

PLI_INT32 aval
PLI_INT32 bval

struct t_vpi_strengthval

Public Members

PLI_INT32 logic
PLI_INT32 s0
PLI_INT32 s1

struct t_vpi_value

Public Members

PLI_INT32 format
PLI_BYTE8 *str
PLI_INT32 scalar
PLI_INT32 integer
double real
struct t_vpi_time *time
struct t_vpi_vecval *vector
struct t_vpi_strengthval *strength
PLI_BYTE8 *misc
union t_vpi_value::[anonymous] value

struct t_vpi_arrayvalue

Public Members

PLI_UINT32 format
PLI_UINT32 flags
PLI_INT32 *integers
PLI_UINT16 *shortints
PLI_INT64 *longints
PLI_BYTE8 *rawvals
struct t_vpi_vecval *vectors
struct t_vpi_time *times
double *reals
float *shortreals
union t_vpi_arrayvalue::[anonymous]value

struct t_vpi_systf_data

Public Members

PLI_INT32 type
PLI_INT32 sysfunctype
const PLI_BYTE8 *tfname
PLI_INT32 (*calltf)(PLI_BYTE8 *)
PLI_INT32 (*compiletf)(PLI_BYTE8 *)
PLI_INT32 (*sizetf)(PLI_BYTE8 *)
PLI_BYTE8 *user_data

struct t_vpi_vlog_info

Public Members

PLI_INT32 argc
PLI_BYTE8 **argv
PLI_BYTE8 *product
PLI_BYTE8 *version

struct t_vpi_error_info

Public Members

PLI_INT32 state
PLI_INT32 level
PLI_BYTE8 *message
PLI_BYTE8 *product
PLI_BYTE8 *code
PLI_BYTE8 *file
PLI_INT32 line

struct t_cb_data
Public Members

`PLI_INT32 reason`

`PLI_INT32 (*cb_rtn)(struct t_cb_data *)`

`vpiHandle obj`

`p_vpi_time time`

`p_vpi_value value`

`PLI_INT32 index`

`PLI_BYTE8 *user_data`

File `vpi_user_ext.h`

Defines

`vpiRealNet 526`

`vpiInterconnectNet 533`

`vpiInterconnectArray 534`

8.1.3 Struct list

Struct `_log_level_table`

`struct _log_level_table`

Struct `gpi_sim_info_s`

`struct gpi_sim_info_s`

Struct `module_state`

`struct module_state`

Struct `sim_time`

`struct sim_time`
Struct t_callback_data
struct t_callback_data

Struct t_cb_data
struct t_cb_data

Struct t_vpi_arrayvalue
struct t_vpi_arrayvalue

Struct t_vpi_assertion_step_info
struct t_vpi_assertion_step_info

Struct t_vpi_attempt_info
struct t_vpi_attempt_info

Struct t_vpi_delay
struct t_vpi_delay

Struct t_vpi_error_info
struct t_vpi_error_info

Struct t_vpi_strengthval
struct t_vpi_strengthval

Struct t_vpi_systf_data
struct t_vpi_systf_data

Struct t_vpi_time
struct t_vpi_time
Struct t_vpi_value
struct t_vpi_value

Struct t_vpi_vecval
struct t_vpi_vecval

Struct t_vpi_vlog_info
struct t_vpi_vlog_info

Struct vhpiCbDataS
struct vhpiCbDataS

Struct vhpiErrorInfoS
struct vhpiErrorInfoS

Struct vhpiForeignDataS
struct vhpiForeignDataS

Struct vhpiPhysS
struct vhpiPhysS

Struct vhpiTimeS
struct vhpiTimeS

Struct vhpiValueS
struct vhpiValueS
TUTORIAL: ENDIAN SWAPPER

In this tutorial we’ll use some of the built-in features of cocotb to quickly create a complex testbench.

Note: All the code and sample output from this example are available on EDA Playground

For the impatient this tutorial is provided as an example with cocotb. You can run this example from a fresh checkout:

```
cd examples/endian_swapper/tests
make
```

9.1 Design

We have a relatively simplistic RTL block called the endian_swapper. The DUT has three interfaces, all conforming to the Avalon standard:

![Diagram](image)

The DUT will swap the endianness of packets on the Avalon-ST bus if a configuration bit is set. For every packet arriving on the `stream_in` interface the entire packet will be endian swapped if the configuration bit is set, otherwise the entire packet will pass through unmodified.
9.2 Testbench

To begin with we create a class to encapsulate all the common code for the testbench. It is possible to write directed tests without using a testbench class however to encourage code re-use it is good practice to create a distinct class.

```python
class EndianSwapperTB(object):
    def __init__(self, dut):
        self.dut = dut
        self.stream_in = AvalonSTDriver(dut, "stream_in", dut.clk)
        self.stream_out = AvalonSTMonitor(dut, "stream_out", dut.clk)
        self.csr = AvalonMaster(dut, "csr", dut.clk)
        self.expected_output = []
        self.scoreboard = Scoreboard(dut)
        self.scoreboard.add_interface(self.stream_out, self.expected_output)

        # Reconstruct the input transactions from the pins and send them to our 'model'
        self.stream_in_recovered = AvalonSTMonitor(dut, "stream_in", dut.clk,
                                                   callback=self.model)
```

With the above code we have created a testbench with the following structure:

If we inspect this line-by-line:

```python
self.stream_in = AvalonSTDriver(dut, "stream_in", dut.clk)
```

Here we are creating an `AvalonSTDriver` instance. The constructor requires 3 arguments - a handle to the entity containing the interface (`dut`), the name of the interface (`stream_in`) and the associated clock with which to drive the interface (`dut.clk`). The driver will auto-discover the signals for the interface, assuming that they follow the naming convention `<interface_name>_<signal>`. In this case we have the following signals defined for the `stream_in` interface:
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description (from Avalon Specification)</th>
</tr>
</thead>
<tbody>
<tr>
<td>stream_in_data</td>
<td>data</td>
<td>The data signal from the source to the sink</td>
</tr>
<tr>
<td>stream_in_empty</td>
<td>empty</td>
<td>Indicates the number of symbols that are empty during cycles that contain the end of a packet</td>
</tr>
<tr>
<td>stream_in_valid</td>
<td>valid</td>
<td>Asserted by the source to qualify all other source to sink signals</td>
</tr>
<tr>
<td>stream_in_startofpacket</td>
<td>startofpacket</td>
<td>Asserted by the source to mark the beginning of a packet</td>
</tr>
<tr>
<td>stream_in_endofpacket</td>
<td>endofpacket</td>
<td>Asserted by the source to mark the end of a packet</td>
</tr>
<tr>
<td>stream_in_ready</td>
<td>ready</td>
<td>Asserted high to indicate that the sink can accept data</td>
</tr>
</tbody>
</table>

By following the signal naming convention the driver can find the signals associated with this interface automatically.

```python
self.stream_out = AvalonSTMonitor(dut, "stream_out", dut.clk)
self.csr = AvalonMaster(dut, "csr", dut.clk)
```

We do the same to create the monitor on stream_out and the CSR interface.

```python
self.expected_output = []
self.scoreboard = Scoreboard(dut)
self.scoreboard.add_interface(self.stream_out, self.expected_output)
```

The above lines create a `Scoreboard` instance and attach it to the `stream_out` monitor instance. The scoreboard is used to check that the DUT behavior is correct. The call to `add_interface()` takes a Monitor instance as the first argument and the second argument is a mechanism for describing the expected output for that interface. This could be a callable function but in this example a simple list of expected transactions is sufficient.

```python
# Reconstruct the input transactions from the pins and send them to our 'model'
self.stream_in_recovered = AvalonSTMonitor(dut, "stream_in", dut.clk, callback=self._model)
```

Finally we create another Monitor instance, this time connected to the `stream_in` interface. This is to reconstruct the transactions being driven into the DUT. It’s good practice to use a monitor to reconstruct the transactions from the pin interactions rather than snooping them from a higher abstraction layer as we can gain confidence that our drivers and monitors are functioning correctly.

We also pass the keyword argument `callback` to the monitor constructor which will result in the supplied function being called for each transaction seen on the bus with the transaction as the first argument. Our model function is quite straightforward in this case - we simply append the transaction to the expected output list and increment a counter:

```python
def model(self, transaction):
    """Model the DUT based on the input transaction""
    self.expected_output.append(transaction)
    self.pkts_sent += 1
```

### 9.2.1 Test Function

There are various ‘knobs’ we can tweak on this testbench to vary the behavior:

- Packet size
- Backpressure on the `stream_out` interface
- Idle cycles on the `stream_in` interface
- Configuration switching of the endian swap register during the test.
We want to run different variations of tests but they will all have a very similar structure so we create a common `run_test` function. To generate backpressure on the `stream_out` interface we use the `BitDriver` class from `cocotb.drivers`.

```python
@cocotb.coroutine
def run_test(dut, data_in=None, config_coroutine=None, idle_inserter=None, backpressure_inserter=None):
    cocotb.fork(Clock(dut.clk, 5000).start())
    tb = EndianSwapperTB(dut)
    yield tb.reset()
    dut.stream_out_ready <= 1

    # Start off any optional coroutines
    if config_coroutine is not None:
        cocotb.fork(config_coroutine(tb.csr))
    if idle_inserter is not None:
        tb.stream_in.set_valid_generator(idle_inserter())
    if backpressure_inserter is not None:
        tb.backpressure.start(backpressure_inserter())

    # Send in the packets
    for transaction in data_in():
        yield tb.stream_in.send(transaction)

    # Wait at least 2 cycles where output ready is low before ending the test
    for i in range(2):
        yield RisingEdge(dut.clk)
        while not dut.stream_out_ready.value:
            yield RisingEdge(dut.clk)
    pkt_count = yield tb.csr.read(1)

    if pkt_count.integer != tb.pkts_sent:
        raise TestFailure("DUT recorded \$d packets but tb counted \$d" % (pkt_count.integer, tb.pkts_sent))
    else:
        dut._log.info("DUT correctly counted \$d packets" % pkt_count.integer)
    raise tb.scoreboard.result
```

We can see that this test function creates an instance of the testbench, resets the DUT by running the coroutine `tb.reset()` and then starts off any optional coroutines passed in using the keyword arguments. We then send in all the packets from `data_in`, ensure that all the packets have been received by waiting 2 cycles at the end. We read the packet count and compare this with the number of packets. Finally we use the `tb.scoreboard.result` to determine the status of the test. If any transactions didn’t match the expected output then this member would be an instance of the `TestFailure` result.
9.2.2 Test permutations

Having defined a test function we can now auto-generate different permutations of tests using the `TestFactory` class:

```python
factory = TestFactory(run_test)
factory.add_option("data_in", [random_packet_sizes])
factory.add_option("config_coroutine", [None, randomly_switch_config])
factory.add_option("idle_inserter", [None, wave, intermittent_single_cycles, random_50_percent])
factory.add_option("backpressure_inserter", [None, wave, intermittent_single_cycles, random_50_percent])
factory.generate_tests()
```

This will generate 32 tests (named `run_test_001` to `run_test_032`) with all possible permutations of the options provided for each argument. Note that we utilize some of the built-in generators to toggle backpressure and insert idle cycles.
One of the benefits of Python is the ease with which interfacing is possible. In this tutorial we’ll look at interfacing the standard GNU ping command to the simulator. Using Python we can ping our DUT with fewer than 50 lines of code. For the impatient this tutorial is provided as an example with cocotb. You can run this example from a fresh checkout:

```
cd examples/ping_tun_tap/tests
sudo make
```

**Note:** To create a virtual interface the test either needs root permissions or have CAP_NET_ADMIN capability.

### 10.1 Architecture

We have a simple RTL block that takes ICMP echo requests and generates an ICMP echo response. To verify this behavior we want to run the ping utility against our RTL running in the simulator.

In order to achieve this we need to capture the packets that are created by ping, drive them onto the pins of our DUT in simulation, monitor the output of the DUT and send any responses back to the ping process.

Linux has a TUN/TAP virtual network device which we can use for this purpose, allowing ping to run unmodified and unaware that it is communicating with our simulation rather than a remote network endpoint.
First of all we need to work out how to create a virtual interface. Python has a huge developer base and a quick search of the web reveals a TUN example that looks like an ideal starting point for our testbench. Using this example we write a function that will create our virtual interface:

```python
import subprocess, fcntl, struct

def create_tun(name="tun0", ip="192.168.255.1"):
    TUNSETIFF = 0x400454ca
    TUNSETOWNER = TUNSETIFF + 2
    IFF_TUN = 0x0001
    IFF_NO_PI = 0x1000
    tun = open('/dev/net/tun', 'r+b')
    ifr = struct.pack('16sH', name, IFF_TUN | IFF_NO_PI)
    fcntl.ioctl(tun, TUNSETIFF, ifr)
    fcntl.ioctl(tun, TUNSETOWNER, 1000)
    subprocess.check_call('ifconfig tun0 %s up pointopoint 192.168.255.2 up' % ip, shell=True)
    return tun
```

Now we can get started on the actual test. First of all we’ll create a clock signal and connect up the Avalon driver and monitor to the DUT. To help debug the testbench we’ll enable verbose debug on the drivers and monitors by setting the log level to logging.DEBUG.
import cocotb
from cocotb.clock import Clock
from cocotb.drivers.avalon import AvalonSTPkts as AvalonSTDriver
from cocotb.monitors.avalon import AvalonSTPkts as AvalonSTMonitor

@c cocotb.test()
def tun_tap_example_test(dut):
    cocotb.fork(Clock(dut.clk, 5000).start())

    stream_in = AvalonSTDriver(dut, "stream_in", dut.clk)
    stream_out = AvalonSTMonitor(dut, "stream_out", dut.clk)

    # Enable verbose logging on the streaming interfaces
    stream_in.log.setLevel(logging.DEBUG)
    stream_out.log.setLevel(logging.DEBUG)

    # Reset the DUT
    dut._log.debug("Resetting DUT")
    dut.reset_n <= 0
    stream_in.bus.valid <= 0
    yield Timer(10, units='ns')
    yield RisingEdge(dut.clk)
    dut.reset_n <= 1
    dut.stream_out_ready <= 1

    # Create our interface (destroyed at the end of the test)
    tun = create_tun()
    fd = tun.fileno()

    # Kick off a ping...
    subprocess.check_call('ping -c 5 192.168.255.2 &', shell=True)

    # Respond to 5 pings, then quit
    for i in range(5):
        cocotb.log.info("Waiting for packets on tun interface")
        packet = os.read(fd, 2048)
        cocotb.log.info("Received a packet!")
        stream_in.append(packet)
        result = yield stream_out.wait_for_recv()
        os.write(fd, str(result))

That’s it - simple!
10.3 Further work

This example is deliberately simplistic to focus on the fundamentals of interfacing to the simulator using TUN/TAP. As an exercise for the reader a useful addition would be to make the file descriptor non-blocking and spawn out separate coroutines for the monitor / driver, thus decoupling the sending and receiving of packets.
Cocotb was designed to provide a common platform for hardware and software developers to interact. By integrating systems early, ideally at the block level, it’s possible to find bugs earlier in the design process.

For any given component that has a software interface there is typically a software abstraction layer or driver which communicates with the hardware. In this tutorial we will call unmodified production software from our testbench and re-use the code written to configure the entity.

For the impatient this tutorial is provided as an example with cocotb. You can run this example from a fresh checkout:

```
$ cd examples/endian_swapper/tests
$ make MODULE=test_endian_swapper_hal
```

Note: SWIG is required to compile the example

### 11.1 Difficulties with Driver Co-simulation

Co-simulating un-modified production software against a block-level testbench is not trivial – there are a couple of significant obstacles to overcome.

#### 11.1.1 Calling the HAL from a test

Typically the software component (often referred to as a Hardware Abstraction Layer or HAL) is written in C. We need to call this software from our test written in Python. There are multiple ways to call C code from Python, in this tutorial we’ll use SWIG to generate Python bindings for our HAL.

#### 11.1.2 Blocking in the driver

Another difficulty to overcome is the fact that the HAL is expecting to call a low-level function to access the hardware, often something like `ioread32`. We need this call to block while simulation time advances and a value is either read or written on the bus. To achieve this we link the HAL against a C library that provides the low level read/write functions. These functions in turn call into cocotb and perform the relevant access on the DUT.
11.2 Cocotb infrastructure

There are two decorators provided to enable this flow, which are typically used together to achieve the required functionality. The `cocotb.external` decorator turns a normal function that isn’t a coroutine into a blocking coroutine (by running the function in a separate thread). The `cocotb.function` decorator allows a coroutine that consumes simulation time to be called by a thread started with `cocotb.external`. The call sequence looks like this:

```
external()
read()/write()
function()
```

11.3 Implementation

11.3.1 Register Map

The endian swapper has a very simple register map:

<table>
<thead>
<tr>
<th>Byte Offset</th>
<th>Register</th>
<th>Bits</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CONTROL</td>
<td>0</td>
<td>R/W</td>
<td>Enable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>31:1</td>
<td>N/A</td>
<td>Reserved</td>
</tr>
<tr>
<td>4</td>
<td>PACKET_COUNT</td>
<td>31:0</td>
<td>RO</td>
<td>Number of Packets</td>
</tr>
</tbody>
</table>

11.3.2 HAL

To keep things simple we use the same RTL from the Tutorial: Endian Swapper. We write a simplistic HAL which provides the following functions:

```c
endian_swapper_enable(endian_swapper_state_t *state);
endian_swapper_disable(endian_swapper_state_t *state);
endian_swapper_get_count(endian_swapper_state_t *state);
```

These functions call `IORD` and `IOWR` – usually provided by the Altera NIOS framework.
11.3.3 IO Module

This module acts as the bridge between the C HAL and the Python testbench. It exposes the \texttt{IORD} and \texttt{IOWR} calls to link the HAL against, but also provides a Python interface to allow the read/write bindings to be dynamically set (through \texttt{set\_write\_function} and \texttt{set\_read\_function} module functions).

In a more complicated scenario, this could act as an interconnect, dispatching the access to the appropriate driver depending on address decoding, for instance.

11.3.4 Testbench

First of all we set up a clock, create an \textit{Avalon Master} interface and reset the DUT. Then we create two functions that are wrapped with the \texttt{cocotb.function} decorator to be called when the HAL attempts to perform a read or write. These are then passed to the \textit{IO Module}:

```python
@cocotb.function
def read(address):
    master.log.debug("External source: reading address 0x%08X" % address)
    value = yield master.read(address)
    master.log.debug("Reading complete: got value 0x%08x" % value)
    raise ReturnValue(value)

@cocotb.function
def write(address, value):
    master.log.debug("Write called for 0x%08X -> %d" % (address, value))
    yield master.write(address, value)
    master.log.debug("Write complete")

io_module.set_write_function(write)
io_module.set_read_function(read)
```

We can then initialize the HAL and call functions, using the \texttt{cocotb.external} decorator to turn the normal function into a blocking coroutine that we can \texttt{yield}:

```python
state = hal.endian_swapper_init(0)
yield cocotb.external(hal.endian_swapper_enable)(state)
```

The HAL will perform whatever calls it needs, accessing the DUT through the \textit{Avalon-MM driver}, and control will return to the testbench when the function returns.

\textbf{Note:} The decorator is applied to the function before it is called.

11.4 Further Work

You may also consider co-simulating unmodified drivers written using \texttt{mmap} (for example built upon the \texttt{UIO framework}), or interfacing with emulators like \texttt{QEMU} to co-simulate when the software needs to execute on a different processor architecture.
Apart from the examples covered with full tutorials in the previous sections, the directory `cocotb/examples/` contains some more smaller modules you may want to take a look at.

### 12.1 Adder

The directory `cocotb/examples/adder/` contains an adder RTL in both Verilog and VHDL, an `adder_model` implemented in Python, and the cocotb testbench with two defined tests a simple `adder_basic_test()` and a slightly more advanced `adder_randomised_test()`.

This example does not use any `Driver`, `Monitor`, or `Scoreboard`; not even a clock.

### 12.2 D Flip-Flop

The directory `cocotb/examples/dff/` contains a simple D flip-flop, implemented in both VHDL and Verilog.

The HDL has the data input port `d`, the clock port `c`, and the data output `q` with an initial state of 0. No reset port exists.

The cocotb testbench checks the initial state first, then applies random data to the data input. The flip-flop output is captured at each rising edge of the clock and compared to the applied input data using a `Scoreboard`.

The testbench defines a `BitMonitor` (a sub-class of `Monitor`) as a pendant to the cocotb-provided `BitDriver`. The `BitDriver`’s `start()` and `stop()` methods are used to start and stop generation of input data.

A `TestFactory` is used to generate the random tests.

### 12.3 Mean

The directory `cocotb/examples/mean/` contains a module that calculates the mean value of a data input bus `i` (with signals `i_data` and `i_valid`) and outputs it on `o` (with `i_data` and `o_valid`).

It has implementations in both VHDL and SystemVerilog.

The testbench defines a `StreamBusMonitor` (a sub-class of `BusMonitor`), a clock generator, a `value_test` helper coroutine and a few tests. Test `mean_randomised_test` uses the `StreamBusMonitor` to feed a `Scoreboard` with the collected transactions on input bus `i`. 
12.4 Mixed Language

The directory `cocotb/examples/mixed_language/` contains two toplevel HDL files, one in VHDL, one in SystemVerilog, that each instantiate the `endian_swapper` in SystemVerilog and VHDL in parallel and chains them together so that the endianness is swapped twice.

Thus, we end up with SystemVerilog+VHDL instantiated in VHDL and SystemVerilog+VHDL instantiated in SystemVerilog.

The cocotb testbench pulls the reset on both instances and checks that they behave the same.

**Todo:** This example is not complete.

12.5 AXI Lite Slave

The directory `cocotb/examples/axi_lite_slave/` contains...

**Todo:** Write documentation, see README.md

12.6 Sorter

Example testbench for snippet of code from `comp.lang.verilog`:

```python
@cocotb.coroutine
def run_test(dut, data_generator=random_data, delay_cycles=2):
    """Send data through the DUT and check it is sorted output."""
    cocotb.fork(Clock(dut.clk, 100).start())

    # Don't check until valid output
    expected = [None] * delay_cycles

    for index, values in enumerate(data_generator(bits=len(dut.in1))):
        expected.append(sorted(values))

        yield RisingEdge(dut.clk)
        dut.in1 = values[0]
        dut.in2 = values[1]
        dut.in3 = values[2]
        dut.in4 = values[3]
        dut.in5 = values[4]

        yield ReadOnly()
        expect = expected.pop(0)

        if expect is None:
            continue

        got = [int(dut.out5), int(dut.out4), int(dut.out3),
               int(dut.out2), int(dut.out1)]
```

(continues on next page)
```python
if got != expect:
    dut._log.error('Expected $s' % expect)
    dut._log.error('Got $s' % got)
    raise TestFailure("Output didn't match")

dut._log.info('Successfully sent $d cycles of data' % (index + 1))
```
13.1 Simulation Hangs

Did you directly call a function that is decorated as a coroutine, i.e. without using await or yield?

13.2 Increasing Verbosity

If things fail in the VPI/VHPI/FLI area, check your simulator’s documentation to see if it has options to increase its verbosity about what may be wrong. You can then set these options on the make command line as COMPILE_ARGS, SIM_ARGS or EXTRA_ARGS (see Build options and Environment Variables for details).

13.3 Attaching a Debugger

In order to give yourself time to attach a debugger to the simulator process before it starts to run, you can set the environment variable COCOTB_ATTACH to a pause time value in seconds. If set, cocotb will print the process ID (PID) to attach to and wait the specified time before actually letting the simulator run.

For the GNU debugger GDB, the command is attach <process-id>.

This page documents any known quirks and gotchas in the various simulators.

### 14.1 Icarus

#### 14.1.1 Accessing bits in a vector

Accessing bits of a vector doesn’t work:

```plaintext
dut.stream_in_data[2] <= 1
```

See access_single_bit test in examples/functionality/tests/test_discovery.py.

#### 14.1.2 Waveforms

To get waveforms in VCD format some Verilog code must be added to the top component as shown in the example below:

```verilog
module button_deb(
    input clk,
    input rst,
    input button_in,
    output button_valid);

    //... Verilog module code here

    // the "macro" to dump signals
    ifdef COCOTB_SIM
        initial begin
            $dumpfile("button_deb.vcd");
            $dumpvars (0, button_deb);
            #1;
        end
    endif
    endmodule
```
14.2 Verilator

cocotb supports Verilator 4.020 and above. Verilator converts Verilog code to C++ code that is compiled. It does not support VHDL. One major limitation compared to standard Verilog simulators is that it does not support delayed assignments.

To run cocotb with Verilator, you need `verilator` in your PATH.

Finally, cocotb currently generates a Verilator toplevel C++ simulation loop which is timed at the highest precision. If your design’s clocks vary in precision, the performance of the simulation can be improved in the same order of magnitude by adjusting the precision in the Makefile, e.g.,

```
-COCOTB_HDL_TIMEPRECISION = 1us # Set precision to 10^-6s
```

New in version 1.3.

14.3 Synopsys VCS

14.4 Aldec Riviera-PRO

The `LICENSE_QUEUE` environment variable can be used for this simulator – this setting will be mirrored in the TCL `license_queue` variable to control runtime license checkouts.

14.5 Mentor Questa

14.6 Mentor ModelSim

Any ModelSim PE or ModelSim PE derivative (like ModelSim Microsemi, Intel, Lattice Edition) does not support the VHDL FLI feature. If you try to run with FLI enabled, you will see a `vsim-FLI-3155` error:

```
** Error (suppressible): (vsim-FLI-3155) The FLI is not enabled in this version of ModelSim.
```

ModelSim DE and SE (and Questa, of course) supports the FLI.

14.7 Cadence Incisive, Cadence Xcelium

14.8 GHDL

Support is preliminary. Noteworthy is that despite GHDL being a VHDL simulator, it implements the VPI interface.
cocotb is in active development.

We use GitHub issues to track our pending tasks. Take a look at the open Feature List to see the work that’s lined up.

If you have a GitHub account you can also raise an enhancement request to suggest new features.
RELEASE NOTES

All releases are available from the GitHub Releases Page.

16.1 cocotb 1.3.0

Released on 08 January 2020

This will likely be the last release to support Python 2.7.

16.1.1 New features

• Initial support for the Verilator simulator (version 4.020 and above). The integration of Verilator into cocotb is not yet as fast or as powerful as it is for other simulators. Please use the latest version of Verilator, and report bugs if you experience problems.

• New makefile variables `COCOTB_HDL_TIMEUNIT` and `COCOTB_HDL_TIMEPRECISION` for setting the default time unit and precision that should be assumed for simulation when not specified by modules in the design. (#1113)

• New `timeout_time` and `timeout_unit` arguments to `cocotb.test()`, for adding test timeouts. (#1119)

• `cocotb.triggers.with_timeout()`, for a shorthand for waiting for a trigger with a timeout. (#1119)

• The `expect_error` argument to `cocotb.test()` now accepts a specific exception type. (#1116)

• New environment variable `COCOTB_RESULTS_FILE`, to allow configuration of the xunit XML output filename. (#1053)

• A new `bus_separator` argument to `cocotb.drivers.BusDriver`. (#1160)

• A new `start_high` argument to `cocotb.clock.Clock.start()`. (#1036)

• A new `cocotb.__version__` constant, which contains the version number of the running cocotb. (#1196)
16.1.2 Notable changes and bug fixes

- DeprecationWarnings are now shown in the output by default.
- Tracebacks are now preserved correctly for exceptions in Python 2. The tracebacks in all Python versions are now a little shorter.
- `cocotb.external()` and `cocotb.function()` now work more reliably and with fewer race conditions.
- A failing `assert` will be considered a test failure. Previously, it was considered a test `error`.
- `drivers()` and `loads()` now also work correctly in Python 3.7 onwards.
- `cocotb.triggers.Timer` can now be used with `decimal.Decimal` instances, allowing constructs like `Timer(Decimal('1e-9'), units='sec')` as an alternate spelling for `Timer(100, units='us')`. (#1114)
- Many (editorial) documentation improvements.

16.1.3 Deprecations

- `cocotb.result.raise_error` and `cocotb.result.create_error` are deprecated in favor of using Python exceptions directly. `TestError` can still be used if the same exception type is desired. (#1109)
- The `AvalonSTPktsWithChannel` type is deprecated. Use the `report_channel` argument to `AvalonSTPkts` instead.
- The `colour` attribute of log objects like `cocotb.log` or `some_coro.log` is deprecated. Use `cocotb.utils.want_color_output()` instead. (#1231)

16.1.4 Other news

- `cocotb` is now packaged for Fedora Linux and available as `python-cocotb`. (Fedora bug #1747574) (thanks Ben Rosser)

16.2 cocotb 1.2.0

Released on 24 July 2019

16.2.1 New features

- `cocotb` is now built as Python package and installable through pip. (#517, #799, #800, #803, #805)
- Support for `async` functions and generators was added (Python 3 only). Please have a look at `Async functions` for an example how to use this new feature.
- VHDL block statements can be traversed. (#850)
- Support for Python 3.7 was added.
16.2.2 Notable changes and bug fixes

- The heart of cocotb, its scheduler, is now even more robust. Many small bugs, inconsistencies and unreliable behavior have been ironed out.
- Exceptions are now correctly propagated between coroutines, giving users the “natural” behavior they’d expect with exceptions. (#633)
- The `setimmediatevalue()` function now works for values larger than 32 bit. (#768)
- The documentation was cleaned up, improved and extended in various places, making it more consistent and complete.
- Tab completion in newer versions of IPython is fixed. (#825)
- Python 2.6 is officially not supported any more. cocotb supports Python 2.7 and Python 3.5+.
- The cocotb GitHub project moved from potentialventures/cocotb to cocotb/cocotb. Redirects for old URLs are in place.

16.2.3 Deprecations

- The `bits` argument to `BinaryValue`, which is now called `n_bits`.
- The `logger` attribute of log objects like `cocotb.log` or `some_coro.log`, which is now just an alias for `self`.
- The `cocotb.utils.get_python_integer_types` function, which was intended to be private.

16.2.4 Known issues

- Depending on your simulation, cocotb 1.2 might be roughly 20 percent slower than cocotb 1.1. Much of the work in this release cycle went into fixing correctness bugs in the scheduler, sometimes at the cost of performance. We are continuing to investigate this in issue #961. Independent of the cocotb version, we recommend using the latest Python 3 version, which is shown to be significantly faster than previous Python 3 versions, and slightly faster than Python 2.7.

Please have a look at the issue tracker for more outstanding issues and contribution opportunities.

16.3 cocotb 1.1

Released on 24 January 2019.

This release is the result of four years of work with too many bug fixes, improvements and refactoring to name them all. Please have a look at the release announcement on the mailing list for further information.
16.4 cocotb 1.0

Released on 15 February 2015.

16.4.1 New features

• FLI support for ModelSim
• Mixed Language, Verilog and VHDL
• Windows
• 300% performance improvement with VHPI interface
• WaveDrom support for wave diagrams.

16.5 cocotb 0.4

Released on 25 February 2014.

16.5.1 New features

• Issue #101: Implement Lock primitive to support mutex
• Issue #105: Compatibility with Aldec Riviera-Pro
• Issue #109: Combine multiple results.xml into a single results file
• Issue #111: XGMII drivers and monitors added
• Issue #113: Add operators to BinaryValue class
• Issue #116: Native VHDL support by implementing VHPI layer
• Issue #117: Added AXI4-Lite Master BFM

16.5.2 Bugs fixed

• Issue #100: Functional bug in endian_swapper example RTL
• Issue #102: Only 1 coroutine wakes up of multiple coroutines wait() on an Event
• Issue #114: Fix build issues with Cadence IUS simulator

16.5.3 New examples

• Issue #106: TUN/TAP example using ping
16.6 cocotb 0.3

Released on 27 September 2013.
This contains a raft of fixes and feature enhancements.

16.7 cocotb 0.2

Released on 19 July 2013.

16.7.1 New features

• Release 0.2 supports more simulators and increases robustness over 0.1.
• A centralized installation is now supported (see documentation) with supporting libraries build when the simulation is run for the first time.

16.8 cocotb 0.1

Released on 9 July 2013.
• The first release of cocotb.
• Allows installation and running against Icarus, VCS, Aldec simulators.
INDICES AND TABLES

• genindex
• modindex
• search
C

cocotb.handle, 45
cocotb.result, 29
cocotb.scoreboard, 40
cocotb.utils, 42
cocotb.wavedrom, 53
### Symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>attribute</strong></td>
<td>(C macro), 92</td>
</tr>
<tr>
<td>__check_vhpi_error</td>
<td>(C++ function), 80</td>
</tr>
<tr>
<td>__check_vpi_error</td>
<td>(C++ function), 86</td>
</tr>
<tr>
<td>__gpi_get_handle_by_name</td>
<td>(C++ function), 78</td>
</tr>
<tr>
<td>__gpi_get_handle_by_raw</td>
<td>(C++ function), 78</td>
</tr>
<tr>
<td>_driver_send</td>
<td>(cocotb.drivers.BusDriver attribute), 38</td>
</tr>
<tr>
<td>_driver_send()</td>
<td>(cocotb.drivers.Driver method), 37</td>
</tr>
<tr>
<td>_log_level_table</td>
<td>(C++ class), 97, 161</td>
</tr>
<tr>
<td>_log_level_table::level</td>
<td>(C++ member), 97</td>
</tr>
<tr>
<td>_monitor_recv</td>
<td>(cocotb.monitors.Monitor attribute), 40</td>
</tr>
<tr>
<td>_next_valids()</td>
<td>(cocotb.drivers.ValidatedBusDriver method), 39</td>
</tr>
<tr>
<td>_recv()</td>
<td>(cocotb.monitors.Monitor method), 40</td>
</tr>
<tr>
<td>_send</td>
<td>(cocotb.drivers.Driver attribute), 37</td>
</tr>
<tr>
<td>_state</td>
<td>(C++ member), 108</td>
</tr>
<tr>
<td>_wait</td>
<td>(cocotb.triggers.Waitable attribute), 36</td>
</tr>
<tr>
<td>_wait_for_nsignal</td>
<td>(cocotb.drivers.BusDriver attribute), 39</td>
</tr>
<tr>
<td>_wait_for_signal</td>
<td>(cocotb.drivers.BusDriver attribute), 39</td>
</tr>
</tbody>
</table>

### A

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>acquire()</td>
<td>(cocotb.triggers.Lock method), 22</td>
</tr>
<tr>
<td>add()</td>
<td>(cocotb.scheduler.Scheduler method), 54</td>
</tr>
<tr>
<td>add_interface()</td>
<td>(cocotb.scoreboard.Scoreboard method), 41</td>
</tr>
<tr>
<td>add_module_constants</td>
<td>(C++ function), 105</td>
</tr>
<tr>
<td>add_option()</td>
<td>(cocotb.regression.TestFactory method), 32</td>
</tr>
<tr>
<td>add_test()</td>
<td>(cocotb.scheduler.Scheduler method), 54</td>
</tr>
<tr>
<td>append()</td>
<td>(cocotb.drivers.Driver method), 37</td>
</tr>
<tr>
<td>argv</td>
<td>(C++ member), 96</td>
</tr>
<tr>
<td>assign()</td>
<td>(cocotb.binary.BinaryValue method), 33</td>
</tr>
<tr>
<td>AvalonMaster</td>
<td>(class in cocotb.drivers.avalon), 50</td>
</tr>
<tr>
<td>AvalonMemory</td>
<td>(class in cocotb.drivers.avalon), 50</td>
</tr>
<tr>
<td>AvalonMM</td>
<td>(class in cocotb.drivers.avalon), 50</td>
</tr>
<tr>
<td>AvalonST</td>
<td>(class in cocotb.monitors.avalon), 52</td>
</tr>
<tr>
<td>AvalonSTPKts</td>
<td>(class in cocotb.drivers.avalon), 50</td>
</tr>
<tr>
<td>AXI4LiteMaster</td>
<td>(class in cocotb.drivers.amba), 49</td>
</tr>
<tr>
<td>AXI4Slave</td>
<td>(class in cocotb.drivers.amba), 49</td>
</tr>
</tbody>
</table>

### B

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BinaryRepresentation</td>
<td>(class in cocotb.binary), 32</td>
</tr>
<tr>
<td>BinaryValue</td>
<td>(class in cocotb.binary), 32</td>
</tr>
<tr>
<td>binstr()</td>
<td>(cocotb.binary.BinaryValue property), 33</td>
</tr>
<tr>
<td>BitDriver</td>
<td>(class in cocotb.drivers), 38</td>
</tr>
<tr>
<td>buff()</td>
<td>(cocotb.binary.BinaryValue property), 33</td>
</tr>
<tr>
<td>Bus</td>
<td>(class in cocotb.bus), 34</td>
</tr>
<tr>
<td>BusDriver</td>
<td>(class in cocotb.drivers), 38</td>
</tr>
<tr>
<td>BusMonitor</td>
<td>(class in cocotb.monitors), 40</td>
</tr>
</tbody>
</table>

### C

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cache_time</td>
<td>(C++ member), 106</td>
</tr>
<tr>
<td>capture()</td>
<td>(cocotb.bus.Bus method), 34</td>
</tr>
<tr>
<td>CASE_OPTION</td>
<td>(C macro), 77</td>
</tr>
<tr>
<td>CASE_STR</td>
<td>(C macro), 79, 85</td>
</tr>
<tr>
<td>cbAfterDelay</td>
<td>(C macro), 155</td>
</tr>
<tr>
<td>cbAssertionDisable</td>
<td>(C macro), 118</td>
</tr>
<tr>
<td>cbAssertionDisableFailAction</td>
<td>(C macro), 118</td>
</tr>
<tr>
<td>cbAssertionDisablePassAction</td>
<td>(C macro), 118</td>
</tr>
<tr>
<td>cbAssertionDisableVacuousAction</td>
<td>(C macro), 118</td>
</tr>
<tr>
<td>cbAssertionEnable</td>
<td>(C macro), 118</td>
</tr>
<tr>
<td>cbAssertionEnableFailAction</td>
<td>(C macro), 118</td>
</tr>
<tr>
<td>cbAssertionEnableNonvacuousAction</td>
<td>(C macro), 118</td>
</tr>
<tr>
<td>cbAssertionEnablePassAction</td>
<td>(C macro), 118</td>
</tr>
<tr>
<td>cbAssertionFailure</td>
<td>(C macro), 118</td>
</tr>
<tr>
<td>cbAssertionKill</td>
<td>(C macro), 118</td>
</tr>
<tr>
<td>cbAssertionLock</td>
<td>(C macro), 118</td>
</tr>
<tr>
<td>cbAssertionReset</td>
<td>(C macro), 118</td>
</tr>
<tr>
<td>cbAssertionStepFailure</td>
<td>(C macro), 118</td>
</tr>
<tr>
<td>cbAssertionStepSuccess</td>
<td>(C macro), 118</td>
</tr>
<tr>
<td>cbAssertionSuccess</td>
<td>(C macro), 118</td>
</tr>
</tbody>
</table>
cbAssertionSysDisableFailAction
    macro, 118

cbAssertionSysDisablePassAction
    macro, 118

cbAssertionSysDisableVacuousAction
    macro, 118

cbAssertionSysEnableFailAction (C macro), 118

cbAssertionSysEnableNonvacuousAction (C macro), 118

cbAssertionSysEnablePassAction (C macro), 118

cbAssertionSysEnd (C macro), 118

cbAssertionSysInitialized (C macro), 118

cbAssertionSysKill (C macro), 118

cbAssertionSysLock (C macro), 118

cbAssertionSysOff (C macro), 118

cbAssertionSysOn (C macro), 118

cbAssertionSysReset (C macro), 118

cbAssertionSysUnlock (C macro), 118

cbAssertionSysUnlock (C macro), 118

cbAssertionVacuousSuccess (C macro), 118

cbAssign (C macro), 156

cbAtEndOfSimTime (C macro), 156

cbAtStartOfSimTime (C macro), 155

cbCreateObj (C macro), 117

cbDeassign (C macro), 156

cbDisable (C macro), 156

cbEndOfCompile (C macro), 155

cbEndOfFrame (C macro), 117

cbEndOfObject (C macro), 117

cbEndOfObject (C macro), 117

cbEndOfObject (C macro), 117

cbEndOfObject (C macro), 117

cbEndOfObject (C macro), 117

cbEndOfObject (C macro), 117

cbEndOfObject (C macro), 117

cbError (C macro), 155

cbExitInteractive (C macro), 156

ccbForce (C macro), 155

ccbInteractiveScopeChange (C macro), 156

ccnBASynch (C macro), 156

cbNextSimTime (C macro), 155

cbPLIError (C macro), 156

cbReadOnlySynch (C macro), 155

ccRelease (C macro), 155

ccbSignal (C macro), 156

ccSizeChange (C macro), 117

ccStartOfFrame (C macro), 117

ccStartOfReset (C macro), 156

ccStartOfRestart (C macro), 156

(C cbStartOfSave (C macro), 156

(C cbStartOfSimulation (C macro), 155

(C cbStartOfThread (C macro), 117

ccStmt (C macro), 155

ccTchkViolation (C macro), 155

ccUnresolvedSystf (C macro), 156

ccValueChange (C macro), 155

CHECK_AND_STORE (C macro), 77

cccheck_vhpi_error (C macro), 80

cccheck_vpi_error (C macro), 86

cleanup () (cocotb.scheduler.Scheduler method), 55

clear () (cocotb.drivers.Driver method), 37

clear () (cocotb.triggers.Event method), 22

clear () (cocotb.wavedrom.Wavedrom method), 53

clear_log_filter (C++ function), 97, 98

clear_log_handler (C++ function), 97, 98

CLEAR_STORE (C macro), 77

Clock (class in cocotb.clock), 35, 41

ClockCycles (class in cocotb.triggers), 19

cocotb.handle (module), 45

cocotb.result (module), 29

cocotb.scoreboard (module), 40


cocotb.util (module), 42


cocotb.wavedrom (module), 53

COCOTB_ACTIVE_ID (C macro), 106

COCOTB_INACTIVE_ID (C macro), 106

COCOTB_ATTACH, 13, 183


cocotb_entrypoint (C++ class), 66, 92

cocotb_entrypoint::cocotb_arch (C++

class), 66, 92

COCOTB_HDL_TIMEPRECISION

Make Variable, 12

COCOTB_HDL_TIMEUNIT

Make Variable, 12

COCOTB_INACTIVE_ID (C macro), 106

COCOTB_ATTACH, 13, 183


cocotb_entrypoint (C++ class), 66, 92

Combine (class in cocotb.triggers), 21

compare () (cocotb.scoreboard.Scoreboard method), 40

COMPILE_ARGS

Make Variable, 12

ConstantObject (class in cocotb.handle), 46

coroutine (class in cocotb), 30

create_error () (in module cocotb.result), 29

CUSTOM_COMPILE_DEPS

Make Variable, 12

CUSTOM_SIM_DEPS

Make Variable, 12

D
deregister_callback (C++ function), 105, 107
Index 201
FliIterator::FliIterator
FliIterator::~FliIterator
FliIterator
FliIntObjHdl::initialise
FliIntObjHdl::get_signal_value_long
FliIntObjHdl::set_signal_value (C++ function), 73
FliIterator (C++ class), 58, 74
FliIterator::~FliIterator (C++ function), 75
FliIterator::iterate_over (C++ member), 75
FliIterator::m_iterator (C++ member), 75
FliIterator::m_currentHandles (C++ member), 75
FliIterator::populate_handle_list (C++ function), 75
FliIterator::selected (C++ member), 75
FliIterator::m_iterator (C++ member), 75
FliIterator::m_regs (C++ member), 75
FliIterator::mvars (C++ member), 75
FliIterator::next_handle (C++ member), 75
FliIterator::one2many (C++ member), 75
FliIterator::populate_handle_list (C++ function), 75
FliIterator::initialise (C++ function), 72
FliLogicObjHdl (C++ class), 59, 72
FliLogicObjHdl::~FliLogicObjHdl (C++ function), 72
FliLogicObjHdl::FliLogicObjHdl (C++ function), 72
FliLogicObjHdl::get_signal_value_binstr (C++ function), 72
FliLogicObjHdl::initialise (C++ function), 73
FliLogicObjHdl::m_enum_map (C++ member), 73
FliLogicObjHdl::m_mti_buff (C++ member), 73
FliLogicObjHdl::m_num_enum (C++ member), 73
FliLogicObjHdl::m_value_enum (C++ member), 73
FliLogicObjHdl::set_signal_value (C++ function), 72
FliNextPhaseCbHdl (C++ class), 59, 68
FliNextPhaseCbHdl::~FliNextPhaseCbHdl (C++ function), 70
FliNextPhaseCbHdl::FliNextPhaseCbHdl (C++ function), 68
FliNextPhaseCbHdl::run_callback (C++ function), 69
FliObj (C++ class), 59, 70
FliObj::~FliObj (C++ function), 70
FliObj::FliObj (C++ function), 70
FliObj::get_acc_full_type (C++ function), 70
FliObj::get_acc_type (C++ function), 70
FliObj::m_acc_full_type (C++ member), 70
FliObj::m_acc_type (C++ member), 70
FliObjHdl (C++ class), 59, 70
FliObjHdl::~FliObjHdl (C++ function), 70
FliObjHdl::FliObjHdl (C++ function), 70
FliObjHdl::initialise (C++ function), 70
FliProcessCbHdl (C++ class), 59, 67
FliProcessCbHdl::~FliProcessCbHdl (C++ function), 67
FliProcessCbHdl::arm_callback (C++ function), 67
FliProcessCbHdl::cleanup_callback (C++ function), 67
FliProcessCbHdl::FliProcessCbHdl (C++ function), 67
FliProcessCbHdl::m_proc_hdl (C++ member), 67
FliProcessCbHdl::m_sensitised (C++ member), 67
FliReadOnlyCbHdl (C++ class), 59, 68
FliReadOnlyCbHdl::~FliReadOnlyCbHdl (C++ function), 69
FliReadOnlyCbHdl::FliReadOnlyCbHdl (C++ function), 69
FliReadWriteCbHdl (C++ class), 59, 68
FliReadWriteCbHdl::~FliReadWriteCbHdl (C++ function), 68
FliReadWriteCbHdl::FliReadWriteCbHdl (C++ function), 68
FliRealObjHdl (C++ class), 59, 73
FliRealObjHdl::~FliRealObjHdl (C++ function), 73
FliRealObjHdl::FliRealObjHdl (C++ function), 73
FliRealObjHdl::get_signal_value_real (C++ function), 73
FliRealObjHdl::initialise (C++ function), 73
FliRealObjHdl::m_enum_map (C++ member), 73
FliRealObjHdl::m_mti_buff (C++ member), 73
FliRealObjHdl::m_num_enum (C++ member), 73
FliRealObjHdl::m_value_enum (C++ member), 73
FliRealObjHdl::set_signal_value (C++ function), 73
FliShutdownCbHdl (C++ class), 60, 69
FliShutdownCbHdl::~FliShutdownCbHdl (C++ function), 69
FliShutdownCbHdl::arm_callback (C++ function), 69
FliShutdownCbHdl::FliShutdownCbHdl (C++ function), 69
FliShutdownCbHdl::run_callback (C++ function), 69
FliSignalCbHdl (C++ class), 60, 67
FliSignalCbHdl::~FliSignalCbHdl (C++ function), 68
FliSignalCbHdl::arm_callback (C++ function), 67
FliSignalCbHdl::cleanup_callback (C++ function), 68

202 Index
FliSignalCbHdl::FliSignalCbHdl (C++ function), 67
FliSignalCbHdl::m_sig_hdl (C++ member), 68
FliSignalObjHdl (C++ class), 60, 70
FliSignalObjHdl::~FliSignalObjHdl (C++ function), 71
FliSignalObjHdl::FliSignalObjHdl (C++ function), 71
FliSignalObjHdl::initialise (C++ function), 71
FliSignalObjHdl::is_var (C++ function), 71
FliSignalObjHdl::m_either_cb (C++ member), 71
FliSignalObjHdl::m_falling_cb (C++ member), 71
FliSignalObjHdl::m_is_var (C++ member), 71
FliSignalObjHdl::m_rising_cb (C++ member), 71
FliSignalObjHdl::value_change_cb (C++ function), 71
FliSimPhaseCbHdl (C++ class), 60, 68
FliSimPhaseCbHdl::~FliSimPhaseCbHdl (C++ function), 68
FliSimPhaseCbHdl::arm_callback (C++ function), 68
FliSimPhaseCbHdl::FliSimPhaseCbHdl (C++ function), 68
FliSimPhaseCbHdl::m_priority (C++ member), 68
FliStartupCbHdl (C++ class), 60, 69
FliStartupCbHdl::~FliStartupCbHdl (C++ function), 69
FliStartupCbHdl::arm_callback (C++ function), 69
FliStartupCbHdl::FliStartupCbHdl (C++ function), 69
FliStartupCbHdl::run_callback (C++ function), 69
FliStringObjHdl (C++ class), 61, 73
FliStringObjHdl::~FliStringObjHdl (C++ function), 74
FliStringObjHdl::FliStringObjHdl (C++ function), 74
FliStringObjHdl::get_signal_value_str (C++ function), 74
FliStringObjHdl::initialise (C++ function), 74
FliStringObjHdl::m_mti_buff (C++ member), 74
FliStringObjHdl::set_signal_value (C++ function), 74
FliTimedCbHdl (C++ class), 61, 69
FliTimedCbHdl::~FliTimedCbHdl (C++ function), 70
FliTimedCbHdl::arm_callback (C++ function), 70
FliTimedCbHdl::cleanup_callback (C++ function), 70
FliTimedCbHdl::FliTimedCbHdl (C++ function), 70
FliTimedCbHdl::m_time_ps (C++ member), 70
FliTimedCbHdl::reset_time (C++ function), 70
FliTimerCache (C++ class), 61, 74
FliTimerCache::~FliTimerCache (C++ function), 74
FliTimerCache::FliTimerCache (C++ function), 74
FliTimerCache::free_list (C++ member), 74
FliTimerCache::get_timer (C++ function), 74
FliTimerCache::impl (C++ member), 74
FliTimerCache::put_timer (C++ function), 74
FliValueObjHdl (C++ class), 61, 71
FliValueObjHdl::~FliValueObjHdl (C++ function), 71
FliValueObjHdl::FliValueObjHdl (C++ function), 71
FliValueObjHdl::get_fli_typeid (C++ function), 71
FliValueObjHdl::get_fli_typekind (C++ function), 71
FliValueObjHdl::get_signal_value_binstr (C++ function), 71
FliValueObjHdl::get_signal_value_long (C++ function), 71
FliValueObjHdl::get_signal_value_real (C++ function), 71
FliValueObjHdl::get_signal_value_str (C++ function), 71
FliValueObjHdl::get_sub_hdl (C++ function), 71
FliValueObjHdl::initialise (C++ function), 71
FliValueObjHdl::m_fli_type (C++ member), 72
FliValueObjHdl::m_sub_hdls (C++ member), 72
FliValueObjHdl::m_val_buff (C++ member), 72
FliValueObjHdl::m_val_type (C++ member), 72
FliValueObjHdl::set_signal_value (C++ function), 71
fork() (in module cocotb), 35
function (class in cocotb), 31

G
GEN_IDX_SEP_LHS (C macro), 80
GEN_IDX_SEP_RHS (C macro), 80
GpiIterator::m_parent (C++ member), 103
GpiIterator::next_handle (C++ function), 103
GpiIteratorMapping (C++ class), 62, 103
GpiIteratorMapping::add_to_options (C++ function), 103
GpiIteratorMapping::get_options (C++ function), 103
GpiIteratorMapping::GpiIteratorMapping (C++ function), 103
GpiIteratorMapping::options_map (C++ member), 103
GpiObjHdl (C++ class), 62, 99
GpiObjHdl::~GpiObjHdl (C++ function), 100
GpiObjHdl::get_const (C++ function), 100
GpiObjHdl::get_definition_file (C++ function), 100
GpiObjHdl::get_definition_name (C++ function), 100
GpiObjHdl::getfullname (C++ function), 100
GpiObjHdl::getfullname_str (C++ function), 100
GpiObjHdl::get_indexable (C++ function), 100
GpiObjHdl::get_name (C++ function), 100
GpiObjHdl::get_name_str (C++ function), 100
GpiObjHdl::get_num_elems (C++ function), 100
GpiObjHdl::get_range_left (C++ function), 100
GpiObjHdl::get_range_right (C++ function), 100
GpiObjHdl::get_type (C++ function), 100
GpiObjHdl::get_type_str (C++ function), 100
GpiObjHdl::GpiObjHdl (C++ function), 100
GpiObjHdl::initialise (C++ function), 100
GpiObjHdl::m_fullname (C++ member), 100
GpiObjHdl::m_definition_file (C++ member), 100
GpiObjHdl::m_definition_name (C++ member), 100
GpiObjHdl::m_fullname (C++ member), 100
GpiObjHdl::m_indexable (C++ member), 100
GpiObjHdl::m_name (C++ member), 100
GpiObjHdl::m_num elems (C++ member), 100
GpiObjHdl::m_range_left (C++ member), 100
GpiObjHdl::m_range_right (C++ member), 100
GpiObjHdl::m_type (C++ member), 100
GpiSignalObjHdl (C++ class), 62, 100
GpiSignalObjHdl::~GpiSignalObjHdl (C++ function), 101
GpiSignalObjHdl::GpiSignalObjHdl (C++ function), 101
GpiSignalObjHdl::GpiSignalObjHdl::get_signal_value_binstr (C++ function), 101
GpiSignalObjHdl::GpiSignalObjHdl::get_signal_value_long (C++ function), 101
GpiSignalObjHdl::get_signal_value_real (C++ function), 101
GpiSignalObjHdl::get_signal_value_str (C++ function), 101
GpiSignalObjHdl::m_length (C++ member), 101
GpiSignalObjHdl::set_signal_value (C++ function), 101
GpiSignalObjHdl::value_change_cb (C++ function), 101
GPITrigger (class in cocotb.triggers), 36
GpiValueCbHdl (C++ class), 63, 102
GpiValueCbHdl::GpiValueCbHdl (C++ function), 102
GpiValueCbHdl::cleanup_callback (C++ function), 102
GpiValueCbHdl::m_signal (C++ member), 102
GpiValueCbHdl::required_value (C++ member), 102
GpiValueCbHdl::run_callback (C++ function), 102
GPIWarning (C++ enumerator), 98
gtstate (C++ member), 96
GUI
Make Variable, 11
H
handle_fli_callback (C++ function), 66, 67
handle_gpi_callback (C++ function), 105
handle_vhpi_callback (C++ function), 79
handle_vpi_callback (C++ function), 85
hexdiffs () (in module cocotb.utils), 44
hexdump () (in module cocotb.utils), 43
HierarchyArrayObject (class in cocotb.handle), 46
HierarchyObject (class in cocotb.handle), 45
hook (class in cocotb), 31
I
idle () (cocotb.drivers.xgmii.XGMII method), 52
in_reset() (cocotb.monitors.BusMonitor property), 40
INITERROR (C macro), 104
integer () (cocotb.binary.BinaryValue property), 33
IntegerObject (class in cocotb.handle), 48
is_const (C++ function), 79
is_enum_boolean (C++ function), 79
is_enum_logic (C++ function), 79
is_enum_char (C++ function), 79
is_python_context (C++ member), 91
is_resolvable() (cocotb.binary.BinaryValue property), 33
iterate (C++ function), 105, 107

J
Join (class in cocotb.triggers), 21
join() (in module cocotb.decorators.RunningCoroutine), 36

K
kill() (cocotb.drivers.Driver method), 37
kill() (in module cocotb.decorators.RunningCoroutine), 36

L
layer1() (cocotb.drivers.xgmiIXGMI static method), 51
layer_entry_func (C++ type), 98
lazy_property (class in cocotb.utils), 44
LICENSE_QUEUE, 186
loads() (cocotb.handle.NonConstantObject method), 47
local_level (C++ member), 97
Lock (class in cocotb.triggers), 22
locked (cocotb.triggers.Lock attribute), 22
log_buff (C++ member), 97
LOG_CRITICAL (C macro), 97
LOGDEBUG (C macro), 97
LOG_ERROR (C macro), 97
LOG_INFO (C macro), 97
log_level (C++ function), 97, 105, 107
log_msg (C++ function), 105, 107
LOG_SIZE (C macro), 97
LOG_WARN (C macro), 97

M
main (C++ function), 120
main_time (C++ member), 120
Make Variable
COCOTB_HDL_TIMEPRECISION, 12
COCOTB_HDL_TIMEUNIT, 12
COCOTB_NVC_TRACE, 12
COMPILE_ARGS, 12
CUSTOM_COMPILE_DEPS, 12
CUSTOM_SIM_DEPS, 12
EXTRA_ARGS, 12
GUI, 11
PLUSARGS, 12
SIM, 11
SIM_ARGS, 12
SIM_BUILD, 12
VERILOG_SOURCES, 11
VHDL_SOURCES, 11

N
n_bits() (cocotb.binary.BinaryValue property), 34
NATIVE (C++ enumerator), 102
NATIVE_NO_NAME (C++ enumerator), 102
next (C++ function), 105, 107
NextTimeStep (class in cocotb.triggers), 20
NonConstantObject (class in cocotb.handle), 46
NonHierarchyIndexableObject (class in cocotb.handle), 46
NonHierarchyObject (class in cocotb.handle), 46
NOT_NATIVE (C++ enumerator), 102
NOT_NATIVE_NO_NAME (C++ enumerator), 102

O
OneToMany (C++ enum), 75
OPBMaster (class in cocotb.drivers.opb), 51
OTM_CONSTANTS (C++ enumerator), 75
OTM_END (C++ enumerator), 75
OTM_REGIONS (C++ enumerator), 75
OTM_SIGNAL_SUB_ELEMENTS (C++ enumerator), 75
OTM_SIGNALS (C++ enumerator), 75
OTM_VARIABLE_SUB_ELEMENTS (C++ enumerator), 75

P
p_callback_data (C++ type), 106
p_cb_data (C++ type), 157
p_vpi_arrayvalue (C++ type), 157
p_vpi_assertion_step_info (C++ type), 119
p_vpi_attempt_info (C++ type), 119
p_vpi_delay (C++ type), 156
p_vpi_error_info (C++ type), 157
p_vpi_strengthval (C++ type), 156
p_vpi_systf_data (C++ type), 157
p_vpi_time (C++ type), 156
p_vpi_value (C++ type), 157
p_vpi_vecval (C++ type), 156
p_vpi_vlog_info (C++ type), 157
pack () (in module cocotb.utils), 43
ParametrizedSingleton (class in cocotb.utils), 44
pEventFn (C++ member), 96
setimmediatevalue() (cocotb.handle.EnumObject method), 47
setimmediatevalue() (cocotb.handle.IntegerObject method), 48
setimmediatevalue() (cocotb.handle.ModifiableObject method), 47
setimmediatevalue() (cocotb.handle.RealObject method), 47
setimmediatevalue() (cocotb.handle.StringObject method), 48
signed_integer() (cocotb.binary.BinaryValue property), 33
SIGNED_MAGNITUDE (cocotb.binary.BinaryRepresentation attribute), 32

SIM
  Make Variable, 11
SIM_ARGS
  Make Variable, 12
SIM_BUILD
  Make Variable, 12
sim_ending (C++ member), 106
SIM_FAIL (C++ enumerator), 93
sim_finish_cb (C++ member), 66, 80, 86
SIM_INFO (C++ enumerator), 93
sim_init_cb (C++ member), 66, 80, 86
SIM_TEST_FAIL (C++ enumerator), 93
sim_time (C++ class), 106, 161
sim_time::high (C++ member), 106
sim_time::low (C++ member), 106
sim_to_hdl (C++ function), 99
SimFailure, 29
SimHandle() (in module cocotb.handle), 48
SimHandleBase (class in cocotb.handle), 45
SimTimeoutError, 29
simulator_clear (C++ function), 108
simulator_traverse (C++ function), 108
SimulatorMethods (C++ member), 108
start () (cocotb.clock.Clock attribute), 42
start () (cocotb.drivers.BitDriver method), 38
Status (C++ enum), 102
stop () (cocotb.drivers.BitDriver method), 38
stop_simulator (C++ function), 105, 107
str (C macro), 91
StringObject (class in cocotb.handle), 48
SVPI_TYPES (C macro), 143
system_function_compilef (C++ function), 85
system_function_overload (C++ function), 85
systf_error_level (C++ member), 86
systf_fatal_level (C++ member), 86
systf_info_level (C++ member), 86
systf_warning_level (C++ member), 86

T
  t_callback_data (C++ class), 108, 162
t_callback_data::_saved_thread_state (C++ member), 108
t_callback_data::args (C++ member), 108
t_callback_data::cb_hdl (C++ member), 108
t_callback_data::function (C++ member), 108
t_callback_data::id_value (C++ member), 108
t_callback_data::kwargs (C++ member), 108
t_cb_data (C++ class), 160, 162
t_cb_data::cb_rtn (C++ member), 161
t_cb_data::index (C++ member), 161
t_cb_data::obj (C++ member), 161
t_cb_data::reason (C++ member), 161
t_cb_data::time (C++ member), 161
t_cb_data::user_data (C++ member), 161
t_cb_data::value (C++ member), 161
t_vpi_arrayvalue (C++ class), 159, 162
t_vpi_arrayvalue::flags (C++ member), 159
t_vpi_arrayvalue::format (C++ member), 159
t_vpi_arrayvalue::integers (C++ member), 159
t_vpi_arrayvalue::longints (C++ member), 159
t_vpi_arrayvalue::rawvals (C++ member), 159
t_vpi_arrayvalue::reals (C++ member), 159
t_vpi_arrayvalue::shortints (C++ member), 159
t_vpi_arrayvalue::shortreals (C++ member), 159
t_vpi_arrayvalue::times (C++ member), 159
t_vpi_arrayvalue::value (C++ member), 159
t_vpi_arrayvalue::vectors (C++ member), 159
t_vpi_assertion_step_info (C++ class), 120, 162
t_vpi_assertion_step_info::matched_expression_count (C++ member), 120
t_vpi_assertion_step_info::matched_exprs (C++ member), 120
t_vpi_assertion_step_info::stateFrom (C++ member), 120
t_vpi_assertion_step_info::stateTo (C++ member), 120
t_vpi_attempt_info (C++ class), 120, 162
t_vpi_attempt_info::attemptStartTime (C++ member), 120
t_vpi_attempt_info::detail (C++ member), 120
t_vpi_attempt_info::failExpr (C++ member), 120

Index
vhpiCbHdl::VhpiCbHdl (C++ function), 80
vhpiCbLastKnownDeltaCycle (C macro), 122
vhpiCbNextTimeStep (C macro), 121
vhpiCbPLIError (C macro), 122
vhpiCbQuiescense (C macro), 122
vhpiCbRelease (C macro), 121
vhpiCbRepAfterDelay (C macro), 121
vhpiCbRepEndOfProcesses (C macro), 121
vhpiCbRepEndOfTimeStep (C macro), 122
vhpiCbRepLastKnownDeltaCycle (C macro), 122
vhpiCbRepNextTimeStep (C macro), 121
vhpiCbRepStartOfNextCycle (C macro), 121
vhpiCbRepStartOfPostponed (C macro), 122
vhpiCbRepStartOfProcesses (C macro), 121
vhpiCbResume (C macro), 121
vhpiCbSensitivity (C macro), 122
vhpiCbSigInterrupt (C macro), 122
vhpiCbStartOfAnalysis (C macro), 122
vhpiCbStartOfElaboration (C macro), 122
vhpiCbStartOfInitialization (C macro), 122
vhpiCbStartOfNextCycle (C macro), 121
vhpiCbStartOfPostponed (C macro), 122
vhpiCbStartOfProcesses (C macro), 121
vhpiCbStartOfReset (C macro), 122
vhpiCbStartOfRestart (C macro), 122
vhpiCbStartOfSave (C macro), 122
vhpiCbStartOfSimulation (C macro), 122
vhpiCbStartOfSubpCall (C macro), 121
vhpiCbStartOfTool (C macro), 122
vhpiCbStmt (C macro), 121
vhpiCbSuspend (C macro), 121
vhpiCbTimeOut (C macro), 122
vhpiCbTransaction (C macro), 121
vhpiCbValueChange (C macro), 121
vhpiCharLiteralK (C++ enumerator), 125
vhpiCharT (C++ type), 123
vhpiCharVal (C++ enumerator), 124
vhpiChoices (C++ enumerator), 130
vhpiClassKindT (C++ enum), 124
vhpiCollectionK (C++ enumerator), 127
vhpiComp (C++ enumerator), 138
vhpiCompConfigK (C++ enumerator), 125
vhpiCompDeclK (C++ enumerator), 125
vhpiCompInstKindT (C++ enum), 138
vhpiCompInstStmtK (C++ enumerator), 125
vhpiCompInstStmts (C++ enumerator), 130
vhpiCompNameP (C++ enumerator), 134
vhpiComponentEC (C++ enumerator), 137
vhpiCondExpr (C++ enumerator), 128
vhpiCondExprs (C++ enumerator), 130
vhpiCondSigAssignStmtK (C++ enumerator), 125
vhpiCondWaveformK (C++ enumerator), 125
vhpiCondWaveforms (C++ enumerator), 130
vhpiConfig (C++ enumerator), 138
vhpiConfigDecl (C++ enumerator), 128
vhpiConfigDeclK (C++ enumerator), 125
vhpiConfigItems (C++ enumerator), 130
vhpiConfigSpec (C++ enumerator), 128
vhpiConfigSpecs (C++ enumerator), 130
vhpiConfigurationEC (C++ enumerator), 137
vhpiConnectivity (C++ enumerator), 138
vhpiConstantEC (C++ enumerator), 137
vhpiConstDeclK (C++ enumerator), 125
vhpiConstDecls (C++ enumerator), 130
vhpiConstParamDeclK (C++ enumerator), 125
vhpiConstraint (C++ enumerator), 128
vhpiConstraints (C++ enumerator), 130
vhpiContributor (C++ enumerator), 128
vhpiContributors (C++ enumerator), 130
vhpiConvFuncK (C++ enumerator), 125
vhpiCurCallback (C++ enumerator), 128
vhpiCurEqProcess (C++ enumerator), 128
vhpiCurRegion (C++ enumerator), 130
vhpiCurRegions (C++ enumerator), 132
vhpiCurStackFrame (C++ enumerator), 128
vhpiDecl (C++ enumerator), 128
vhpiDeclS (C++ enumerator), 130
vhpiDecStrVal (C++ enumerator), 123
vhpiDefNameP (C++ enumerator), 134
vhpiDelayedPA (C++ enumerator), 136
vhpiDelayModeT (C++ enum), 139
vhpiDeposit (C++ enumerator), 139
vhpiDepositPropagate (C++ enumerator), 139
vhpiDepUnits (C++ enumerator), 131
vhpiDerefObj (C++ enumerator), 128
vhpiDerefObjK (C++ enumerator), 125
vhpiDesignUnit (C++ enumerator), 128
vhpiDesignUnits (C++ enumerator), 131
vhpiDirect (C++ enumerator), 138
vhpiDisable (C++ enumerator), 138
vhpiDisableCb (C macro), 122
vhpiDisconnectSpecK (C++ enumerator), 125
vhpiDontCare (C macro), 121
vhpiDownStack (C++ enumerator), 128
vhpiDrivenSigs (C++ enumerator), 131
vhpiDriverCollectionK (C++ enumerator), 125
vhpiDriverK (C++ enumerator), 125
vhpiDrivers (C++ enumerator), 131
vhpiDriving_valuePA (C++ enumerator), 136
vhpiDrivingPA (C++ enumerator), 136
vhpiDynamic (C++ enumerator), 136
vhpiEdifUnit (C++ enumerator), 127
vhpiElaborationPhase (C++ enumerator), 138
vhpiElemAssocK (C++ enumerator), 125
vhpiElemAssocs (C++ enumerator), 131
vhpiElemDeclK (C++ enumerator), 125
vhpiElemSubtype (C++ enumerator), 128
VhpiImpl::iterate_handle (C++ function), 84
VhpiImpl::m_next_phase (C++ member), 85
VhpiImpl::m_read_only (C++ member), 85
VhpiImpl::m_read_write (C++ member), 85
VhpiImpl::native_check_create (C++ function), 84
VhpiImpl::reason_to_string (C++ function), 84
VhpiImpl::register_nexttime_callback (C++ function), 84
VhpiImpl::register_readonly_callback (C++ function), 84
VhpiImpl::registerReadWriteCallback (C++ function), 84
VhpiImpl::register_readwrite_callback (C++ function), 84
VhpiImpl::register_timed_callback (C++ function), 84
VhpiImpl::sim_end (C++ function), 84
VhpiImpl::VhpiImpl (C++ function), 84
VhpiIndexedNameK (C++ enumerator), 126
VhpiIndexedNames (C++ enumerator), 131
VhpiIndexExprs (C++ enumerator), 131
VhpiInertial (C++ enumerator), 139
VhpiInitExpr (C++ enumerator), 129
VhpiInitializationPhase (C++ enumerator), 138
VhpiInMode (C++ enumerator), 136
VhpiInOpen (C++ enumerator), 135
VhpiInOutMode (C++ enumerator), 136
VhpiInPort (C++ enumerator), 129
VhpiInPortK (C++ enumerator), 126
VhpiInstance_namePA (C++ enumerator), 136
VhpiInternal (C++ enumerator), 138
VhpiInternalRegions (C++ enumerator), 131
VhpiIntLiteralK (C++ enumerator), 126
VhpiIntPropertyT (C++ enum), 132
VhpiIntRangeK (C++ enumerator), 126
VhpiIntT (C++ type), 123
VhpiIntTypeDec1K (C++ enumerator), 126
VhpiIntPtrVal (C++ enumerator), 124
VhpiIntPtrValP (C++ enumerator), 132
VhpiIntVecVal (C++ enumerator), 124
VhpiIsAnonymousFP (C++ enumerator), 132
VhpiIsBasicP (C++ enumerator), 132
VhpiIsCompositeP (C++ enumerator), 132
VhpiIsDefaultP (C++ enumerator), 132
VhpiIsDeferredP (C++ enumerator), 132
VhpiIsDiscreteP (C++ enumerator), 132
VhpiIsForcedP (C++ enumerator), 132
VhpiIsForeignP (C++ enumerator), 132
VhpiIsGuardedP (C++ enumerator), 132
VhpiIsImplicitDec1P (C++ enumerator), 132
VhpiIsInvalidP (C++ enumerator), 132
VhpiIsLocalP (C++ enumerator), 132
VhpiIsNamedP (C++ enumerator), 132
VhpiIsNullP (C++ enumerator), 132
VhpiIsOpenP (C++ enumerator), 133
VhpiIsPassiveP (C++ enumerator), 133
VhpiIsPrivIP (C++ enumerator), 133
VhpiIsPostponedP (C++ enumerator), 133
VhpiIsProtectedTypeP (C++ enumerator), 133
VhpiIsPureP (C++ enumerator), 133
VhpiIsResolvedP (C++ enumerator), 133
VhpiIsScalarP (C++ enumerator), 133
VhpiIsSequenceP (C++ enumerator), 133
VhpiIsSharedP (C++ enumerator), 133
VhpiIsStdLogicP (C++ enumerator), 134
VhpiIsStdLogicVectorP (C++ enumerator), 134
VhpiIsStdULogicP (C++ enumerator), 134
VhpiIsStdULogicVectorP (C++ enumerator), 134
VhpiIsTransportP (C++ enumerator), 133
VhpiIsUnaffectedP (C++ enumerator), 133
VhpiIsUnconstrainedP (C++ enumerator), 133
VhpiIsUninstantiatedP (C++ enumerator), 133
VhpiIsUpP (C++ enumerator), 133
VhpiIsVitalP (C++ enumerator), 133
VhpiIterator (C++ class), 63, 83
VhpiIterator::~VhpiIterator (C++ function), 84
VhpiIterator::iterate_over (C++ member), 84
VhpiIterator::m_iter_obj (C++ member), 84
VhpiIterator::next_handle (C++ function), 84
VhpiIterator::one2many (C++ member), 84
VhpiIterator::selected (C++ member), 84
VhpiIterator::VhpiIterator (C++ function), 84
VhpiIterator::iterate_over (C++ member), 84
VhpiLanguageP (C++ enumerator), 134
VhpiLast_activePA (C++ enumerator), 136
VhpiLast_eventPA (C++ enumerator), 136
VhpiLast_valuePA (C++ enumerator), 136
VhpiLeftBoundP (C++ enumerator), 133
VhpiLeftExpr (C++ enumerator), 129
VhpiLeftOfPA (C++ enumerator), 136
VhpiLeftOfP (C++ enumerator), 136
VhpiLengthP (C++ enumerator), 136
VhpiLevelP (C++ enumerator), 133
VhpiLexicalScope (C++ enumerator), 129
VhpiLhsExpr (C++ enumerator), 129
VhpiLibP (C++ enumerator), 139
vhpiLibLogicalNameP (C++ enumerator), 134
vhpiLibPhysicalNameP (C++ enumerator), 134
vhpiLibraryDeclK (C++ enumerator), 126
vhpiLibraryDecls (C++ enumerator), 132
vhpiLineOp (C++ enumerator), 133
vhpiLineOffsetP (C++ enumerator), 133
vhpiLinkageMode (C++ enumerator), 136
vhpiLiteralEC (C++ enumerator), 137
vhpiLocal (C++ enumerator), 129
vhpiLocalLoads (C++ enumerator), 132
vhpiLocallyStatic (C++ enumerator), 136
vhpiLogicalExpr (C++ enumerator), 129
vhpiLogicalNameP (C++ enumerator), 134
VhpiLogicSignalObjHdl (C++ class), 63, 83
VhpiLogicSignalObjHdl::VhpiLogicSignalObjHdl (C++ function), 83
VhpiLogicSignalObjHdl::VhpiLogicSignalObjHdl::operator= (C++ function), 83
VhpiLogicSignalObjHdl::set_signal_value (C++ function), 83
VhpiLogicSignalObjHdl::VhpiLogicSignalObjHdl::initialise (C++ function), 83
VhpiLogicSignalObjHdl::VhpiLogicSignalObjHdl::set_signal_value (C++ function), 83
VhpiLogicVal (C++ enumerator), 124
VhpiLogicVecVal (C++ enumerator), 124
VhpiLongIntT (C++ type), 123
VhpiLongIntVal (C++ enumerator), 124
VhpiLongIntVecVal (C++ enumerator), 124
VhpiLoopIndexP (C++ enumerator), 133
VhpiLoopLabelNameP (C++ enumerator), 134
VhpiLoopStmtK (C++ enumerator), 126
VhpiLowPA (C++ enumerator), 136
VhpiMature (C++ enumerator), 138
VhpiMembers (C++ enumerator), 131
VhpiModeP (C++ enumerator), 133
VhpiModeT (C++ enum), 135
VhpiName (C++ enumerator), 129
VhpiNameP (C++ enumerator), 134
VhpiNextPhaseCbHdl (C++ class), 63, 81
VhpiNextPhaseCbHdl::VhpiNextPhaseCbHdl (C++ function), 81
VhpiNextPhaseCbHdl::VhpiNextPhaseCbHdl (C++ function), 81
VhpiNextStmtK (C++ enumerator), 126
VhpiNoAccess (C++ enumerator), 138
VhpiNoActivity (C macro), 123
VhpiNormal (C++ enumerator), 136
VhpiNote (C++ enumerator), 138
VhpiNullLiteralK (C++ enumerator), 126
VhpiNullStmtK (C++ enumerator), 126
VhpiNumDimensionsP (C++ enumerator), 133
VhpiNumFieldsP (C++ enumerator), 133
VhpiNumGenersP (C++ enumerator), 133
VhpiNumLiteralsP (C++ enumerator), 133
VhpiNumMembersP (C++ enumerator), 133
VhpiNumParamsP (C++ enumerator), 133
VhpiNumPortsP (C++ enumerator), 133
VhpiObjHdl (C++ class), 63, 82
VhpiObjHdl::VhpiObjHdl (C++ function), 82
VhpiObjHdl::VhpiObjHdl::initialise (C++ function), 82
VhpiObjHdl::VhpiObjHdl::operator= (C++ function), 82
VhpiObjT (C++ enumerator), 124
VhpiOctStrVal (C++ enumerator), 123
VhpiOneToManyT (C++ enum), 130
VhpiOneToManyT (C++ enum), 128
VhpiOpenModeP (C++ enumerator), 133
VhpiOpenModeT (C++ enum), 135
VhpiOperator (C++ enumerator), 129
VhpiOperatorK (C++ enumerator), 126
VhpiOptimizedLoads (C++ enumerator), 132
VhpiOthers (C++ enumerator), 129
VhpiOthersK (C++ enumerator), 126
VhpiOutMode (C++ enumerator), 136
VhpiOutOpen (C++ enumerator), 135
VhpiOutPort (C++ enumerator), 129
VhpiOutPortK (C++ enumerator), 126
VhpiPackageEC (C++ enumerator), 137
VhpiPackBodyK (C++ enumerator), 126
VhpiPackDeclK (C++ enumerator), 126
VhpiPackInstK (C++ enumerator), 126
VhpiPackInsts (C++ enumerator), 131
VhpiParamAssocs (C++ enumerator), 131
VhpiParamAttrNameK (C++ enumerator), 126
VhpiParamDecl (C++ enumerator), 129
VhpiParamDeclK (C++ enumerator), 133
VhpiParamDeclK (C++ enumerator), 129
VhpiParamExpr (C++ enumerator), 129
VhpiParent (C++ enumerator), 129
VhpiPath_namePA (C++ enumerator), 136
VhpiPhaseP (C++ enumerator), 133
VhpiPhaseT (C++ enum), 138
VhpiPhysLeftBoundP (C++ enumerator), 135
VhpiPhysLiteral (C++ enumerator), 129
VhpiPhysLiteralK (C++ enumerator), 126
VhpiPhysPositionP (C++ enumerator), 135
VhpiPhysPropertyT (C++ enum), 135
VhpiPhysRangeK (C++ enumerator), 126
VhpiPhysRightBoundP (C++ enumerator), 135
VhpiPhys::high (C++ member), 141
VhpiPhys::low (C++ member), 141
VhpiPhys::high (C++ type), 123
VhpiPhysTypeDeclK (C++ enumerator), 126
VhpiPhysVal (C++ enumerators), 124
VhpiPhysValK (C++ enumerators), 135
VhpiPhysVecVal (C++ enumerator), 124
VhpiPortAssocs (C++ enumerators), 131
VhpiPortDecl (C++ enumerator), 126
VhpiPortDeclK (C++ enumerator), 131
vhpiPositionP (C++ enumerator), 133
vhpiPosPA (C++ enumerator), 136
vhpiPrecisionP (C++ enumerator), 135
vhpiPredefAttrP (C++ enumerator), 133
vhpiPredefAttrT (C++ enum), 136
vhpiPredPA (C++ enumerator), 137
vhpiPrefix (C++ enumerator), 129
vhpiPrimaryUnit (C++ enumerator), 129
vhpiProcCallStmtK (C++ enumerator), 126
vhpiProcDeclK (C++ enumerator), 126
vhpiProcedureEC (C++ enumerator), 137
vhpiProcessStmtK (C++ enumerator), 126
vhpiProcF (C++ enumerator), 139
vhpiProcFK (C++ enumerator), 139
vhpiProtectedTypeBody (C++ enumerator), 129
vhpiProtectedTypeBodyK (C++ enumerator), 127
vhpiProtectedTypeDecl (C++ enumerator), 129
vhpiProtectedTypeDecl1K (C++ enumerator), 127
vhpiProtectedTypeK (C++ enumerator), 126
vhpiProvidesAdvancedDebugRuntime (C++ enumerator), 135
vhpiProvidesAdvancedForeignModel (C++ enumerator), 135
vhpiProvidesConnectivity (C++ enumerator), 135
vhpiProvidesDebugRuntime (C++ enumerator), 135
vhpiProvidesDynamicElab (C++ enumerator), 135
vhpiProvidesForeignModel (C++ enumerator), 135
vhpiProvidesHierarchy (C++ enumerator), 135
vhpiProvidesPostAnalysis (C++ enumerator), 135
vhpiProvidesReset (C++ enumerator), 135
vhpiProvidesSaveRestart (C++ enumerator), 135
vhpiProvidesStaticAccess (C++ enumerator), 135
vhpiPtrVal (C++ enumerator), 124
vhpiPtrVecVal (C++ enumerator), 124
vhpiPutValueModeT (C++ enum), 129
vhpiQualifiedExprK (C++ enumerator), 127
vhpiQuietPA (C++ enumerator), 137
vhpiRangeAK (C++ enumerator), 137
vhpiRangePA (C++ enumerator), 137
vhpiRawDataVal (C++ enumerator), 124
vhpiRead (C++ enumerator), 138
VhpiReadOnlyCbHdl (C++ class), 63, 82
VhpiReadOnlyCbHdl::VhpiReadOnlyCbHdl (C++ function), 81
VhpiReadOnlyCbHdl::VhpiReadOnlyCbHdl (C++ function), 81
vhpiReadOpen (C++ enumerator), 135
vhpiSeverityExpr (C++ enumerator), 129
vhpiSeverityT (C++ enum), 138
VhpiShutdownCbHdl (C++ class), 64, 82
VhpiShutdownCbHdl::run_callback (C++ function), 82
VhpiShutdownCbHdl::cleanup_callback (C++ function), 82
VhpiShutdownCbHdl::run_callback (C++ function), 82
VhpiSignalObjHdl::m_value (C++ function), 83
VhpiSignalObjHdl::m_value (C++ function), 83
VhpiSignalObjHdl::m_value (C++ function), 83
VhpiSignalObjHdl::m_value (C++ function), 83
VhpiSignalObjHdl::m_value (C++ function), 83
VhpiSignalObjHdl::set_signal_value (C++ function), 83
VhpiSignalObjHdl::m_rising_cb (C++ member), 83
VhpiSignalObjHdl::m_value (C++ member), 83
VhpiSignalObjHdl::set_signal_value (C++ function), 83
VhpiSignalObjHdl::value_change_cb (C++ function), 83
VhpiSignalObjHdl::VhpiSignalObjHdl (C++ function), 83
vhpiSignals (C++ enumerator), 131
vhpiSigAttributes (C++ enumerator), 131
vhpiSigDeclK (C++ enumerator), 127
vhpiSigDeclsl (C++ enumerator), 131
vhpiSigKindP (C++ enumerator), 134
vhpiSignalAK (C++ enumerator), 137
vhpiSignalEC (C++ enumerator), 137
VhpiSignalObjHdl (C++ class), 64, 82
VhpiSignalObjHdl::VhpiSignalObjHdl (C++ function), 83
VhpiSignalObjHdl::chr2vhpi (C++ function), 83
VhpiSignalObjHdl::get_signal_value_binst (C++ function), 83
VhpiSignalObjHdl::get_signal_value_long (C++ function), 83
VhpiSignalObjHdl::get_signal_value_real (C++ function), 83
VhpiSignalObjHdl::get_signal_value_str (C++ function), 83
VhpiSignalObjHdl::initialise (C++ function), 83
VhpiSignalObjHdl::m_binvalue (C++ member), 83
VhpiSignalObjHdl::m_either_cb (C++ member), 83
VhpiSignalObjHdl::m_falling_cb (C++ member), 83
VhpiSignalObjHdl::m_rising_cb (C++ member), 83
VhpiSignalObjHdl::m_value (C++ member), 83
VhpiSignalObjHdl::set_signal_value (C++ function), 83
VhpiSignalObjHdl::value_change_cb (C++ function), 83
VhpiSignalObjHdl::VhpiSignalObjHdl (C++ function), 83
vhpiSimpleName (C++ enumerator), 129
vhpiSimpleSigAssignStmtK (C++ enumerator), 127
vhpiSimTimeUnitP (C++ enumerator), 135
vhpiSimulationPhase (C++ enumerator), 138
vhpiSizeConstraint (C++ enumerator), 139
vhpiSizeP (C++ enumerator), 134
vhpiSliceNameK (C++ enumerator), 127
vhpiSmallEnumT (C++ type), 123
vhpiSmallEnumVal (C++ enumerator), 124
vhpiSmallEnumVal (C++ enumerator), 124
vhpiSmallPhysT (C++ type), 123
vhpiSmallPhysVal (C++ enumerator), 124
vhpiSmallPhysVal (C++ enumerator), 124
vhpiSpecNames (C++ enumerator), 131
vhpiSpecs (C++ enumerator), 131
vhpiStablePA (C++ enumerator), 137
vhpiStartLineNoP (C++ enumerator), 134
VhpiStartupCbHdl (C++ class), 64, 81
VhpiStartupCbHdl::VhpiStartupCbHdl (C++ function), 82
VhpiStartupCbHdl::cleanup_callback (C++ function), 82
VhpiStartupCbHdl::run_callback (C++ function), 82
VhpiStartupCbHdl::run_callback (C++ function), 82
VhpiStartupCbHdl::run_callback (C++ function), 82
VhpiSimTimeUnitP (C++ enumerator), 135
vhpiSystem (C++ enumerator), 128
vhpiSystemC (C++ enumerator), 128
vhpiSystemS (C++ enumerator), 131
VhpiTimedCbHdl (C++ class), 64, 81
VhpiTimedCbHdl::VhpiTimedCbHdl (C++ function), 81
VhpiTimedCbHdl::cleanup_callback (C++ function), 81
VhpiTimedCbHdl::VhpiTimedCbHdl (C++ function), 81
VhpiTimeExpr (C++ enumerator), 130
VhpiTimeOutExpr (C++ enumerator), 130
VhpiTimeS::C++ class), 141, 163
VhpiTimeS::high (C++ member), 141
VhpiTimeS::low (C++ member), 141
VhpiTimeT (C++ type), 123
VhpiTimeVal (C++ enumerator), 124
VhpiTimeVecVal (C++ enumerator), 124
VhpiTool (C++ enumerator), 130
VhpiToolK (C++ enumerator), 127
VhpiToolVersionP (C++ enumerator), 134
VhpiTransactionK (C++ enumerator), 127
VhpiTransactionPA (C++ enumerator), 137
VhpiTransactions (C++ enumerator), 131
VhpiTransport (C++ enumerator), 139
VhpiTrue (C macro), 121
VhpiType (C++ enumerator), 130
VhpiTypeAK (C++ enumerator), 137
VhpiTypeConvK (C++ enumerator), 127
VhpiTypeEC (C++ enumerator), 137
VhpiTypeMark (C++ enumerator), 130
VhpiTypeMarks (C++ enumerator), 131
VhpiTypes (C++ enumerator), 132
VhpiTypespec (C++ enumerator), 130
VhpiU (C macro), 121
VhpiUnaryExprK (C++ enumerator), 127
VhpiUndefined (C macro), 121
VhpiUnitDecl (C++ enumerator), 130
VhpiUnitDeclK (C++ enumerator), 127
VhpiUnitDecls (C++ enumerator), 131
VhpiUnitNameP (C++ enumerator), 134
VhpiUnitsEC (C++ enumerator), 137
VhpiUpperRegion (C++ enumerator), 130
VhpiUpStack (C++ enumerator), 130
VhpiUse (C++ enumerator), 130
VhpiUseClauseK (C++ enumerator), 127
VhpiUseClauses (C++ enumerator), 132
VhpiUserAttrNameK (C++ enumerator), 127
VhpiUserFctT (C++ type), 123
VhpiUses (C++ enumerator), 131
VhpiValExpr (C++ enumerator), 130
VhpiValPA (C++ enumerator), 137
VhpiValSubtype (C++ enumerator), 130
VhpiValType (C++ enumerator), 130
VhpiValueAK (C++ enumerator), 137
VhpiValueCbHdl (C++ class), 64, 80
VhpiValueCbHdl::VhpiValueCbHdl (C++ function), 81
VhpiValueCbHdl::initial_value (C++ member), 81
VhpiValueCbHdl::rising (C++ member), 81
VhpiValueCbHdl::signal (C++ member), 81
VhpiValueCbHdl::VhpiValueCbHdl (C++ function), 81
VhpiValuePA (C++ enumerator), 137
VhpiValueS (C++ class), 141, 163
VhpiValueS::bufSize (C++ member), 142
VhpiValueS::ch (C++ member), 142
VhpiValueS::enumv (C++ member), 142
VhpiValueS::enumvs (C++ member), 142
VhpiValueS::format (C++ member), 142
VhpiValueS::intg (C++ member), 142
VhpiValueS::intgs (C++ member), 142
VhpiValueS::longintg (C++ member), 142
VhpiValueS::longintgs (C++ member), 142
VhpiValueS::numElems (C++ member), 142
VhpiValueS::phys (C++ member), 142
VhpiValueS::physs (C++ member), 142
VhpiValueS::ptr (C++ member), 142
VhpiValueS::ptrs (C++ member), 142
VhpiValueS::real (C++ member), 142
VhpiValueS::reals (C++ member), 142
VhpiValueS::smallenumv (C++ member), 142
VhpiValueS::smallenumvs (C++ member), 142
VhpiValueS::smallphys (C++ member), 142
VhpiValueS::smallphyss (C++ member), 142
VhpiValueS::str (C++ member), 142
VhpiValueS::time (C++ member), 142
VhpiValueS::times (C++ member), 142
VhpiValueS::unit (C++ member), 142
VhpiValueS::value (C++ member), 142
VhpiValueT (C++ type), 123
VhpiVarAssignStmtK (C++ enumerator), 127
VhpiVarDeclK (C++ enumerator), 127
VhpiVarDecs (C++ enumerator), 132
VhpiVariableEC (C++ enumerator), 137
VhpiVarParamDeclK (C++ enumerator), 127
VhpiVerilog (C++ enumerator), 127
VhpiVHDL (C++ enumerator), 128
VhpiVHDLversionP (C++ enumerator), 134
VhpiW (C macro), 121
VhpiWaitStmtK (C++ enumerator), 127
VhpiWarning (C++ enumerator), 138
VhpiWaveformElemK (C++ enumerator), 127
VhpiWaveformElems (C++ enumerator), 132
VhpiWhileLoopK (C++ enumerator), 127
VhpiWrite (C++ enumerator), 138
VhpiWriteOpen (C++ enumerator), 135
VhpiX (C macro), 121
VhpiZ (C macro), 121
vlog_startup_routines (C++ member), 86
vpiIf (C macro), 144
vpiIfElse (C macro), 144
vpiIfElseOp (C macro), 116
vpiIfOp (C macro), 116
vpiImmediateAssert (C macro), 111
vpiImmediateAssume (C macro), 111
vpiImmediateCover (C macro), 111
VpiImpl (C++ class), 64, 90
VpiImpl::create_gpi_obj_from_handle (C++ function), 90
VpiImpl::deregsister_callback (C++ function), 90
VpiImpl::get_root_handle (C++ function), 90
VpiImpl::get_sim_precision (C++ function), 90
VpiImpl::get_sim_time (C++ function), 90
VpiImpl::iterate_handle (C++ function), 90
VpiImpl::m_next_phase (C++ member), 91
VpiImpl::m_read_only (C++ member), 91
VpiImpl::m_read_write (C++ member), 91
VpiImpl::native_check_create (C++ function), 90
VpiImpl::next_handle (C++ function), 90
VpiImpl::register_nexttime_callback (C++ function), 90
VpiImpl::register_readonly_callback (C++ function), 90
VpiImpl::register_readwrite_callback (C++ function), 90
VpiImpl::register_timed_callback (C++ function), 90
VpiImpl::sim_end (C++ function), 90
VpiImpl::VpiImpl (C++ function), 90
VpiImplication (C macro), 112
VpiImplicitDecl (C macro), 149
VpiImplyOp (C macro), 116
VpiImport (C macro), 115
vpiIndex (C macro), 146
vpiIndexedPartSelect (C macro), 146
vpiIndexedPartSelectType (C macro), 153
vpiIndexTypespec (C macro), 112
vpiInertialDelay (C macro), 154
vpiInitial (C macro), 144
vpiInout (C macro), 148
vpiInput (C macro), 148
vpiInputEdge (C macro), 115
vpiInputSkew (C macro), 112
vpiInsideOp (C macro), 116
vpiInsideQualifier (C macro), 115
vpiInstance (C macro), 113
vpiInstanceArray (C macro), 147
vpiIntConst (C macro), 152
vpiIntegerNet (C macro), 111
vpiIntegerTypespec (C macro), 110
vpiIntegerVar (C macro), 144
vpiInterconnectArray (C macro), 161
vpiInterconnectNet (C macro), 161
vpiInterface (C macro), 109
vpiInterfaceArray (C macro), 109
vpiInterfaceDecl (C macro), 112
vpiInterfacePort (C macro), 113
vpiInterfaceTfDecl (C macro), 109
vpiInTerm (C macro), 147
vpiInterModPath (C macro), 144
vpiInterModPathDelay (C macro), 153
vpiInternal (C macro), 155
vpiInternalScope (C macro), 146
vpiIntersectOp (C macro), 116
vpiIntFunc (C macro), 152
vpiIntTypespec (C macro), 109
vpiIntVal (C macro), 154
vpiIntVar (C macro), 109
vpiIODec1 (C macro), 144
vpiIsClockInferred (C macro), 114
vpiIsConstraintEnabled (C macro), 114
vpiIsCoverSequence (C macro), 114
vpiIsDeferred (C macro), 114
vpiIsFinal (C macro), 114
vpiIsMemory (C macro), 153
vpiIsProtected (C macro), 153
vpiIsRandomized (C macro), 113
vpiIterator (C macro), 144
vpiIterator (C++ class), 65, 89
vpiIterator::~VpiIterator (C++ function), 89
vpiIterator::iterate_over (C++ member), 89
vpiIterator::m_iterator (C++ member), 89
vpiIterator::next_handle (C++ function), 89
vpiIterator::one2many (C++ member), 89
vpiIterator::selected (C++ member), 89
vpiIterator::VpiIterator (C++ function), 89
vpiIteratorType (C macro), 153
vpiJoin (C macro), 113
vpiJoinAny (C macro), 113
vpiJoinNone (C macro), 113
vpiJoinType (C macro), 113
vpiL (C macro), 155
vpiLargeCharge (C macro), 154
vpiLeftRange (C macro), 146
vpiLeOp (C macro), 151
vpiLetDecl (C macro), 112
vpiLetExpr (C macro), 112
vpiLhs (C macro), 146
vpiLibrary (C macro), 153
vpiLineNo (C macro), 147
vpiListOp (C macro), 152
vpiLoad (C macro), 146
vpiLocalDriver (C macro), 147
vpiLocalLoad (C macro), 147
vpiLocalParam (C macro), 153
vpiLocalVarDecls (C macro), 113
vpiLocalVis (C macro), 114
vpiLogAndOp (C macro), 151
vpiLogicNet (C macro), 111
vpiLogicTypespec (C macro), 110
vpiLogicVar (C macro), 109
vpiLogOrOp (C macro), 151
vpiLongIntTypespec (C macro), 109
vpiLongIntVar (C macro), 109
vpiLongIntVal (C macro), 154
vpiLongOrOp (C macro), 151
vpiMatchedOp (C macro), 151
vpiMatchOp (C macro), 113
vpiMatchedOp (C macro), 116
vpiMatchOp (C macro), 116
vpiMediumCharge (C macro), 154
vpiMember (C macro), 113
vpiMemory (C macro), 144
vpiMemoryWord (C macro), 144
vpiMessages (C macro), 112
vpiMethod (C macro), 114
vpiMethodFuncCall (C macro), 110
vpiMethods (C macro), 112
vpiMethodTaskCall (C macro), 110
vpiMinTypMaxOp (C macro), 152
vpiMinusOp (C macro), 151
vpiMIPDelay (C macro), 153
vpiMixedIO (C macro), 148
vpiModDataPathIn (C macro), 146
vpiModel1364v1995 (C macro), 115
vpiModel1364v2001 (C macro), 115
vpiModel1364v2005 (C macro), 115
vpiModel1800v2005 (C macro), 115
vpiModel1800v2009 (C macro), 115
vpiModOp (C macro), 151
vpiModPath (C macro), 144
vpiModPathDelay (C macro), 153
vpiModPathHasIfNone (C macro), 153
vpiModPathIn (C macro), 146
vpiModPathOut (C macro), 146
vpiModPort (C macro), 109
vpiModportPort (C macro), 113
vpiModule (C macro), 144
vpiModuleArray (C macro), 145
vpiMultiArray (C macro), 153
vpiMultiAssignmentPatternOp (C macro), 116
vpiMulticlockSequenceExpr (C macro), 111
vpiMultiConcatOp (C macro), 151
vpiMulOp (C macro), 151
vpiName (C macro), 147
vpiNamed (C macro), 144
vpiNamedBegin (C macro), 144
vpiNamedEvent (C macro), 144
vpiNamedEventArray (C macro), 146
vpiNamedFork (C macro), 144
vpiNandPrim (C macro), 149
vpiNegative (C macro), 150
vpiNegedge (C macro), 150
vpiNegedgeOp (C macro), 152
vpiNegIndexed (C macro), 153
vpiNeqOp (C macro), 151
vpiNet (C macro), 144
vpiNetArray (C macro), 145
vpiNetBit (C macro), 144
vpiNetDeclAssign (C macro), 152
vpiNetType (C macro), 148
vpiNextPhaseCbHdl (C++ class), 65, 87
vpiNextPhaseCbHdl::VpiNextPhaseCbHdl (C++ function), 87
vpiNextPhaseCbHdl::VpiNextPhaseCbHdl (C++ function), 87
vpiNexttimeOp (C macro), 116
vpiNmosPrim (C macro), 149
vpiNoChange (C macro), 150
vpiNoDelay (C macro), 154
vpiNoDirection (C macro), 148
vpiNoEdge (C macro), 150
vpiNone (C macro), 148
vpiNonOverlapFollowedByOp (C macro), 115
vpiNonOverlapImlyOp (C macro), 115
vpiNoQualifier (C macro), 114
vpiNotRand (C macro), 149
vpiNotifyPrIm (C macro), 149
vpiNotifyOp (C macro), 149
vpiNotOp (C macro), 151
vpiNotPrim (C macro), 149
vpiNotRand (C macro), 153
vpiNullConst (C macro), 114
vpiNullOp (C macro), 152
vpiNullStmt (C macro), 144
vpiObjHdl (C++ class), 65, 88
vpiObjHdl::~VpiObjHdl (C++ function), 88
vpiObjHdl::initialise (C++ function), 88
vpiObjHdl::VpiObjHdl (C++ function), 88
vpiObjId (C macro), 115
vpiObjTypeVal (C macro), 154
vpiOctConst (C macro), 152
vpiOctStrVal (C macro), 154
vpiObjId (C macro), 153
vpiOffset (C macro), 153
vpiOneStepConst (C macro), 114
vpiOneValue (C macro), 155
index
<table>
<thead>
<tr>
<th>Function</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>VpiStartupCbHdl::run_callback</td>
<td>C++</td>
</tr>
<tr>
<td>VpiStartupCbHdl::VpiStartupCbHdl function</td>
<td>C++</td>
</tr>
<tr>
<td>vpiStatementCoverage</td>
<td>C macro</td>
</tr>
<tr>
<td>vpiStaticArray</td>
<td>C macro</td>
</tr>
<tr>
<td>vpiStmt</td>
<td>C macro</td>
</tr>
<tr>
<td>vpiStop</td>
<td>C macro</td>
</tr>
<tr>
<td>vpiStreamLRop</td>
<td>C macro</td>
</tr>
<tr>
<td>vpiStreamRLOp</td>
<td>C macro</td>
</tr>
<tr>
<td>vpiStrength0</td>
<td>C macro</td>
</tr>
<tr>
<td>vpiStrength1</td>
<td>C macro</td>
</tr>
<tr>
<td>vpiStrengthVal</td>
<td>C macro</td>
</tr>
<tr>
<td>vpiStringConst</td>
<td>C macro</td>
</tr>
<tr>
<td>vpiStringTypespec</td>
<td>C macro</td>
</tr>
<tr>
<td>vpiStringVal</td>
<td>C macro</td>
</tr>
<tr>
<td>vpiStringVar</td>
<td>C macro</td>
</tr>
<tr>
<td>vpiStringVar</td>
<td>C macro</td>
</tr>
<tr>
<td>vpiStructPattern</td>
<td>C macro</td>
</tr>
<tr>
<td>vpiStructMember</td>
<td>C macro</td>
</tr>
<tr>
<td>vpiStructNet</td>
<td>C macro</td>
</tr>
<tr>
<td>vpiStructPattern</td>
<td>C macro</td>
</tr>
<tr>
<td>vpiStructSpec</td>
<td>C macro</td>
</tr>
<tr>
<td>vpiStructVar</td>
<td>C macro</td>
</tr>
<tr>
<td>vpiSubOp</td>
<td>C macro</td>
</tr>
<tr>
<td>vpiSupply0</td>
<td>C macro</td>
</tr>
<tr>
<td>vpiSupply1</td>
<td>C macro</td>
</tr>
<tr>
<td>vpiSupplyDrive</td>
<td>C macro</td>
</tr>
<tr>
<td>vpiSuppressTime</td>
<td>C macro</td>
</tr>
<tr>
<td>vpiSuppressTime</td>
<td>C macro</td>
</tr>
<tr>
<td>vpiSwitch</td>
<td>C macro</td>
</tr>
<tr>
<td>vpiSwitchArray</td>
<td>C macro</td>
</tr>
<tr>
<td>vpiSyncAcceptOnOp</td>
<td>C macro</td>
</tr>
<tr>
<td>vpiSyncRejectOnOp</td>
<td>C macro</td>
</tr>
<tr>
<td>vpiSysFunc</td>
<td>C macro</td>
</tr>
<tr>
<td>vpiSysFuncCall</td>
<td>C macro</td>
</tr>
<tr>
<td>vpiSysFuncInt</td>
<td>C macro</td>
</tr>
<tr>
<td>vpiSysFuncReal</td>
<td>C macro</td>
</tr>
<tr>
<td>vpiSysFuncSized</td>
<td>C macro</td>
</tr>
<tr>
<td>vpiSysFuncTime</td>
<td>C macro</td>
</tr>
<tr>
<td>vpiSysFuncType</td>
<td>C macro</td>
</tr>
<tr>
<td>vpiSysTask</td>
<td>C macro</td>
</tr>
<tr>
<td>vpiSysTaskCall</td>
<td>C macro</td>
</tr>
<tr>
<td>vpiSystem</td>
<td>C macro</td>
</tr>
<tr>
<td>vpiSysTfCall</td>
<td>C macro</td>
</tr>
<tr>
<td>vpiTableEntry</td>
<td>C macro</td>
</tr>
<tr>
<td>vpiTagged</td>
<td>C macro</td>
</tr>
<tr>
<td>vpiTaggedPattern</td>
<td>C macro</td>
</tr>
<tr>
<td>vpiTaggedQualifier</td>
<td>C macro</td>
</tr>
<tr>
<td>vpiTask</td>
<td>C macro</td>
</tr>
<tr>
<td>vpiTaskCall</td>
<td>C macro</td>
</tr>
<tr>
<td>vpiTaskFunc</td>
<td>C macro</td>
</tr>
<tr>
<td>vpiTchk</td>
<td>C macro</td>
</tr>
<tr>
<td>vpiTchkDataTerm</td>
<td>C macro</td>
</tr>
<tr>
<td>vpiTchkNotifier</td>
<td>C macro</td>
</tr>
</tbody>
</table>
vpiUnboundedConst (C macro), 114
vpiUnconnDrive (C macro), 147
vpiUndefined (C macro), 147
vpiUnionTypespec (C macro), 110
vpiUnionVar (C macro), 109
vpiUniqueQualifier (C macro), 114
vpiUnique (C macro), 113
vpiUnknown (C macro), 150
vpiUntilOp (C macro), 116
vpiUntilWithOp (C macro), 116
vpiUse (C macro), 147
vpiUserAllocFlag (C macro), 155
vpiUserDefinedClass (C macro), 114
vpiUserDefn (C macro), 152
vpiUserSystf (C macro), 145
vpiUserWire (C macro), 148
vpiValid (C macro), 153
vpiValidFalse (C macro), 153
vpiValidTrue (C macro), 153
vpiValidUnknown (C macro), 117
VpiValueCbHdl (C++ class), 66, 86
VpiValueCbHdl::VpiValueCbHdl (C++ function), 87
VpiValueCbHdl::cleanup_callback (C++ function), 87
VpiValueCbHdl::m_vpi_value (C++ member), 87
VpiValueCbHdl::VpiValueCbHdl (C++ function), 87
vpiValueRange (C macro), 112
vpiVarBit (C macro), 109
vpiVariables (C macro), 147
vpiVarSelect (C macro), 145
vpiVector (C macro), 148
vpiVectorVal (C macro), 154
vpiVirtual (C macro), 114
vpiVirtualInterfaceVar (C macro), 109
vpiVisibility (C macro), 114
vpiVoidTypespec (C macro), 110
vpiWait (C macro), 145
vpiWaitFork (C macro), 111
vpiWaitingProcesses (C macro), 112
vpiWand (C macro), 148
vpiWarning (C macro), 155
vpiWeakDrive (C macro), 154
vpiWeight (C macro), 112
vpiWhile (C macro), 145
vpiWidth (C macro), 150
vpiWidthExpr (C macro), 147
vpiWildEqOp (C macro), 116
vpiWildNeqOp (C macro), 116
VpiWire (C macro), 148
vpiWith (C macro), 112
vpiWithinOp (C macro), 116
vpiWor (C macro), 148
vpiX (C macro), 155
vpiXnorPrim (C macro), 149
vpiXorPrim (C macro), 149
vpiZ (C macro), 155

W
wait() (cocotb.triggers.Event method), 22
wait_for_recv() (cocotb.monitors.Monitor method), 39
Waitable (class in cocotb.triggers), 36
want_color_output() (in module cocotb.utils), 44
Wavedrom (class in cocotb.wavedrom), 53
WAVES

Make Variable, 11
with_timeout() (in module cocotb.triggers), 22
write() (cocotb.drivers.amba.AXI4LiteMaster method), 49
write() (cocotb.drivers.avalon.AvalonMaster method), 50
write() (cocotb.drivers.opb.OPBMaster method), 51

X
XGMII (class in cocotb.drivers.xgmii), 51
XGMII (class in cocotb.monitors.xgmii), 52
xstr (C macro), 91
XXTERN (C macro), 120, 143