cocotbext Wishbone

April 1, 2025

Jay Convertino

Contents

1 Usage

1.1 Introduction

Cocotb extension to test Wishbone Classic Standard bus master, and slave devices. This currently only tests for devices that are not pipelined or using cti/bte (registered). Also no tags.

1.2 Dependencies

The following are the dependencies of the cores.

- iverilog (simulation)
- cocotb (simulation)
- cocotb-bus (simulation)

1.3 In a Simulation

Below is a simple example for reading and writing data from register zero in the cocotb extension.

```
master  = wishboneStandardMaster(dut, "s_wb", dut.clk,
    ↪ dut.rst)
slave = wishboneStandardEchoSlave(dut, "s_wb", dut.clk,
    ↪ dut.rst)

await master.write(0, 0xAAAAAAAA)

rx_data = await master.read(0)

assert 0xAAAAAAAA == rx_data, "RECEIVED␣DATA␣DOES␣NOT␣
    ↪ MATCH"
```

2 Architecture

Please see 4 for more information.

wishboneStandardMaster tests Wishbone Classic slave devices by executing read/write requests from the python test bench.

wishboneStandardEchoSlave provides a simple slave that will echo all register writes back over read when requested.

wishboneStandardMonitor tests to make sure signals are proper. Simple core at the moment, only checks for if stb is asserted when cyc is not.

2.1 Directory Guide

Below highlights important folders from the root of the directory.

1. **docs** Contains all documentation related to this project.
   - **manual** Contains user manual and github page that are generated from the latex sources.
2. **cocotbext** Contains source files for the extension
   - **wishbone.standard** Contains source files for the Wishbone version B4 classic standard extension.
3. **tests** Contains test files for cocotb

3 Simulation

A simulation for testing the cores can be run to verify operation.

3.1 cocotb

To use the cocotb tests you must install the following python libraries.

```
$ pip install cocotb
$ pip install −e .
```

Then you must enter the tests folder and enter the mil-std-1553 folder. From there you may execute the following command which will kick off the test.

```
$ make
```

4 Code Documentation

Natural docs is used to generate documentation for this project. The next lists the following sections.

- **init** Python init code.
- **monitor** Contains bus monitor code.
- **driver** Contains bus driver code.
- **absbus** Contains bus abstraction for monitor, and driver code.
- **busbase** Contains bus base for threads and read/write methods.
- **cocotb test** Python TestFactory code.
- **cocotb verilog test wrapper** Verilog wrapper module.

__init__.py

AUTHORS

JAY CONVERTINO

DATES

2025/03/31

INFORMATION

Brief

Wishbone Classic Standard define for packages

License MIT

monitor.py

AUTHORS

JAY CONVERTINO

DATES

2025/03/11

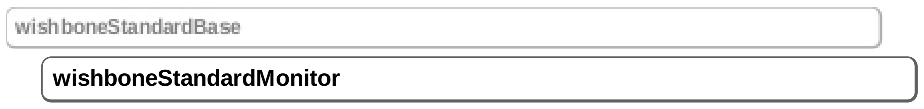
INFORMATION

Brief

Monitor for Wishbone Classic Standard

License MIT

wishboneStandardMonitor



Check signals to make sure they are applied properly.

FUNCTIONS

__init__

```
def __init__(
self,
entity,
name,
clock,
resetn,
                                                                    *
args,
                                                                   **
kwargs
)
```

Setup defaults and call base class constructor.

_check_type

```
def _check_type(
self,
trans
)
```

Check and make sure we are only sending wishboneStandardTrans, this is only here to satisify the need to have it.

_run

```
async def _run(
self
)
```

_run thread that deals with checking signals, simple check for now.

driver.py

AUTHORS

JAY CONVERTINO

DATES

2025/03/11

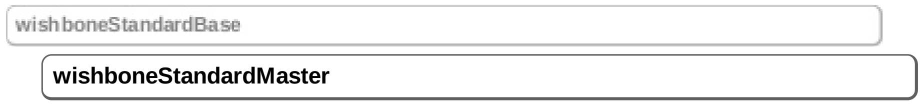
INFORMATION

Brief

Bus Driver for Wishbone Classic Master/echoSlave

License MIT

wishboneStandardMaster



Drive slave devices over the Wishbone Classic bus

FUNCTIONS

__init__

```
def __init__(
self,
entity,
name,
clock,
reset,
                                                                        *
args,
                                                                       **
kwargs
)
```

Setup defaults and call base class constructor.

read

```
async def read(
self,
address
)
```

Read from a address and return data

write

```
async def write(
self,
address,
data
)
```

Write to a address some data

_check_type

```
def _check_type(
self,
trans
)
```

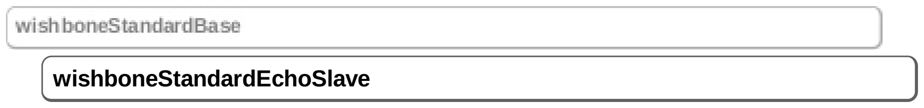
Check and make sure we are only sending 2 bytes at a time and that it is a bytes/bytearray

_run

```
async def _run(
self
)
```

_run thread that deals with read and write queues.

wishboneStandardEchoSlave



Respond to master reads and write by returning data, simple echo core.

FUNCTIONS

__init__

```
def __init__(
self,
entity,
name,
clock,
reset,
numreg
=
256,
                                                                    *
args,
                                                                   **
kwargs
)
```

Setup defaults and call base class constructor.

_check_type

```
def _check_type(
self,
trans
)
```

Check and make sure we are only sending a type of wishboneStandardTrans.

_run

```
async def _run(
self
)
```

_run thread that deals with read and write request over bus.

absbus.py

AUTHORS

JAY CONVERTINO

DATES

2025/03/11

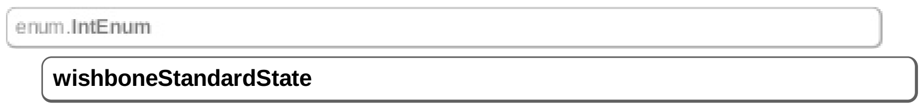
INFORMATION

Brief

abstraction of the wishbone classic standard bus

License MIT

wishboneStandardState



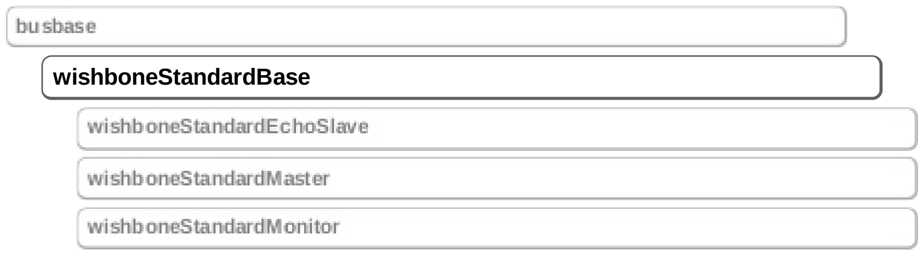
An enum class that provides the current operation state.

wishboneStandardTrans

transaction

Create an object that associates data, address

wishboneStandardBase



abstract base class that defines Wishbone Classic signals

VARIABLES

_signals

```
_signals
```

List of signals that are required

_optional_signals

```
_optional_signals
```

List of optional signals, these will never be required but will be used if found.

FUNCTIONS

__init__

```
def __init__(
self,
entity,
name,
clock,
reset,
                                                                          *
args,
                                                                         **
kwargs
)
```

Setup defaults and call base class constructor.

busbase.py

AUTHORS

JAY CONVERTINO

DATES

2025/03/11

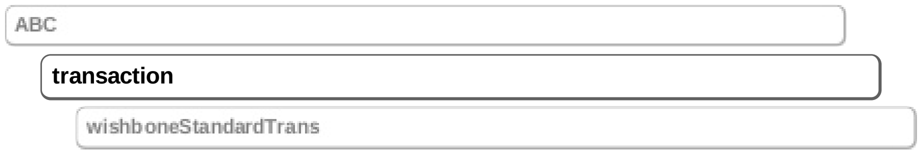
INFORMATION

Brief

classic bus define for packages

License MIT

transaction



Abstract class for transaction types

noSignal

noSignal

Class to use when a signal does not exist

busbase

busbase

wishboneStandardBase

A busbase to transmit test routine.

FUNCTIONS

__init__

```
def __init__(
self,
entity
:
SimHandleBase,
name
:
Optional[str],
clock
:
SimHandleBase,
*
args
:
Any,
**
kwargs
:
Any
)
```

Initialize the object

VARIABLES

wqueue

```
self.wqueue
```

Queue to store write requests

qqueue

```
self.qqueue
```

Queue to store read requests

rqueue

```
self.rqueue
```

Queue to store result of read requests

self._idle

```
self._idle
```

Event trigger for cocotb

self._run_cr

```
self._run_cr
```

Thread instance of _run method

FUNCTIONS

_restart

```
def _restart(
self
)
```

kill and restart _run thread.

write_count

```
def write_count(
self
)
```

How many items in the write queue

read_count

```
def read_count(
self
)
```

How many items in the read queue

write_empty

```
def write_empty(
self
)
```

Is the quene empty?

read_empty

```
def read_empty(
self
)
```

Is the quene empty?

write_clear

```
def write_clear(
self
)
```

Remove all write items from queue

read_clear

```
def read_clear(
self
)
```

Remove all read items from queue

wait

```
async def wait(
self
)
```

Wait for the run thread to become idle.

idle

```
def idle(
self
)
```

Are all the queues empty and the _run is not active processing data.

write_trans

```
async def write_trans(
self,
trans
:
transaction
)
```

Write transaction to send to write queue

read_trans

```
async def read_trans(
self,
trans
:
transaction
)
```

Read bus and output and tranaction.

_write

```
async def _write(
self,
trans
:
transaction
)
```

Write data one element at a time

_queue_read

```
async def _queue_read(
self,
trans
:
transaction
)
```

Setup queue for read requests

_read

```
async def _read(
self,
trans
:
transaction
)
```

Read dat one element at a time

_check_type

```
def _check_type(
self,
trans
)
```

Check and make sure we are only sending the correct transaction type

_run

```
async def _run(
self
)
```

Virtual method for _run thread that deals with read and write queues.

TB

TB

Create the device under test which is the master/slave.

FUNCTIONS

run_test

```
async def run_test(
dut,
payload_data
 =
None
)
```

Tests the source/sink for valid transmission of data.

incrementing_payload

```
def incrementing_payload()
```

Generate a list of ints that increment from 0 to 2^8

test

```
def test(
request
)
```

Main cocotb function that specifies how to put the test together.

test.v

AUTHORS

JAY CONVERTINO

DATES

2025/03/17

INFORMATION

Brief

Test bench for apb using cocotb

License MIT

test

```
module test #(
parameter
ADDRESS_WIDTH
 =
16,
parameter
BUS_WIDTH
 =
4
) ( input clk, input rst, inout s_wb_cyc, inout s_wb_stb, inout s_wb_we, in
```

Test of Wishbone Classic

Parameters

ADDRESS_WIDTH parameter	Width of the Wishbone address port in bits.
BUS_WIDTH parameter	Width of the Wishbone bus data port in bytes.

Ports

clk	Clock
rst	Positive reset
s_wb_cyc	Bus Cycle in process
s_wb_stb	Valid data transfer cycle
s_wb_we	Active High write, low read
s_wb_addr	Bus address
s_wb_data_i	Input data
s_wb_sel	Device Select
s_wb_ack	Bus transaction terminated
s_wb_data_o	Output data
s_wb_err	Active high when a bus error is present