

busbase.py

AUTHORS

JAY CONVERTINO

DATES

2025/03/11

INFORMATION

Brief

classic bus define for packages

License MIT

Copyright 2025 Jay Convertino

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

transaction

ABC

transaction

basictans

Abstract class for transaction types

noSignal

noSignal

Class to use when a signal does not exist

busbase

busbase

basicMaster

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

_idle_read

```
self._idle_read
```

Event trigger for cocotb read

_idle_write

```
self._idle_write
```

Event trigger for cocotb write

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 queue empty?

read_empty

```
def read_empty(  
    self  
)
```

Is the queue empty?self.bus.penable.value

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_read

```
async def wait_read(  
    self  
)
```

Wait for the run thread to become idle from a read.

wait_write

```
async def wait_write(  
    self
```

```
)
```

Wait for the run thread to become idle from a write.

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 transaction.

_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.