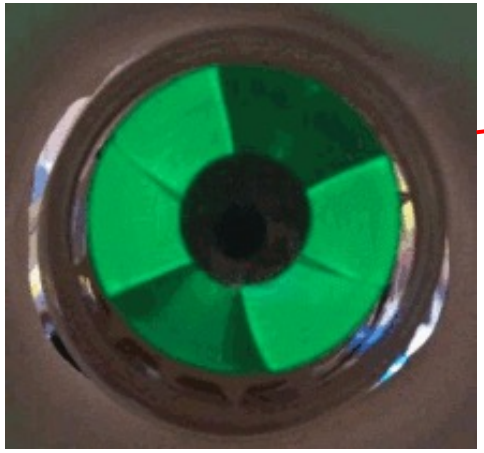
6.S965
Digital Systems Laboratory II

Lecture 6:

Scoreboarding and Models and Drivers

Administrative

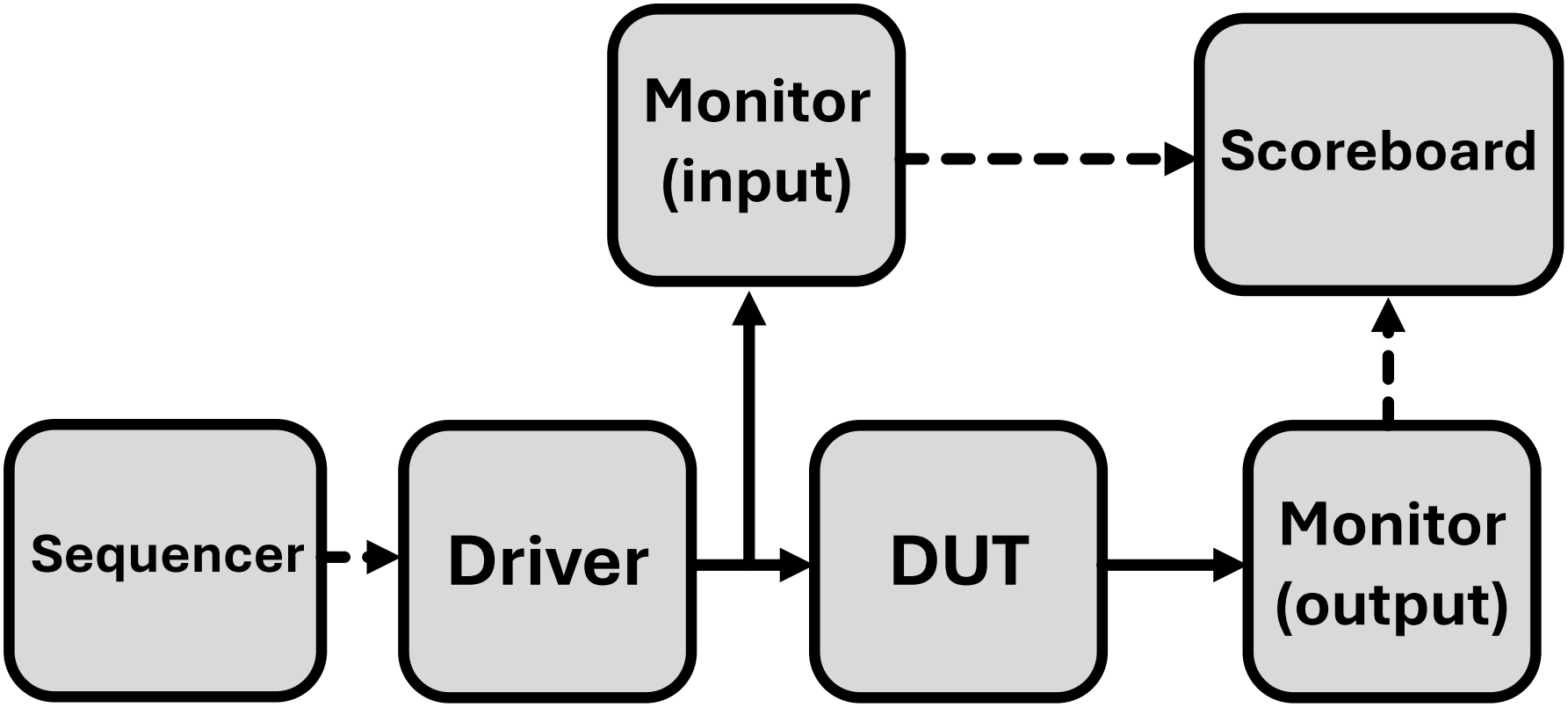
- Week 3 stuff is due tomorrow



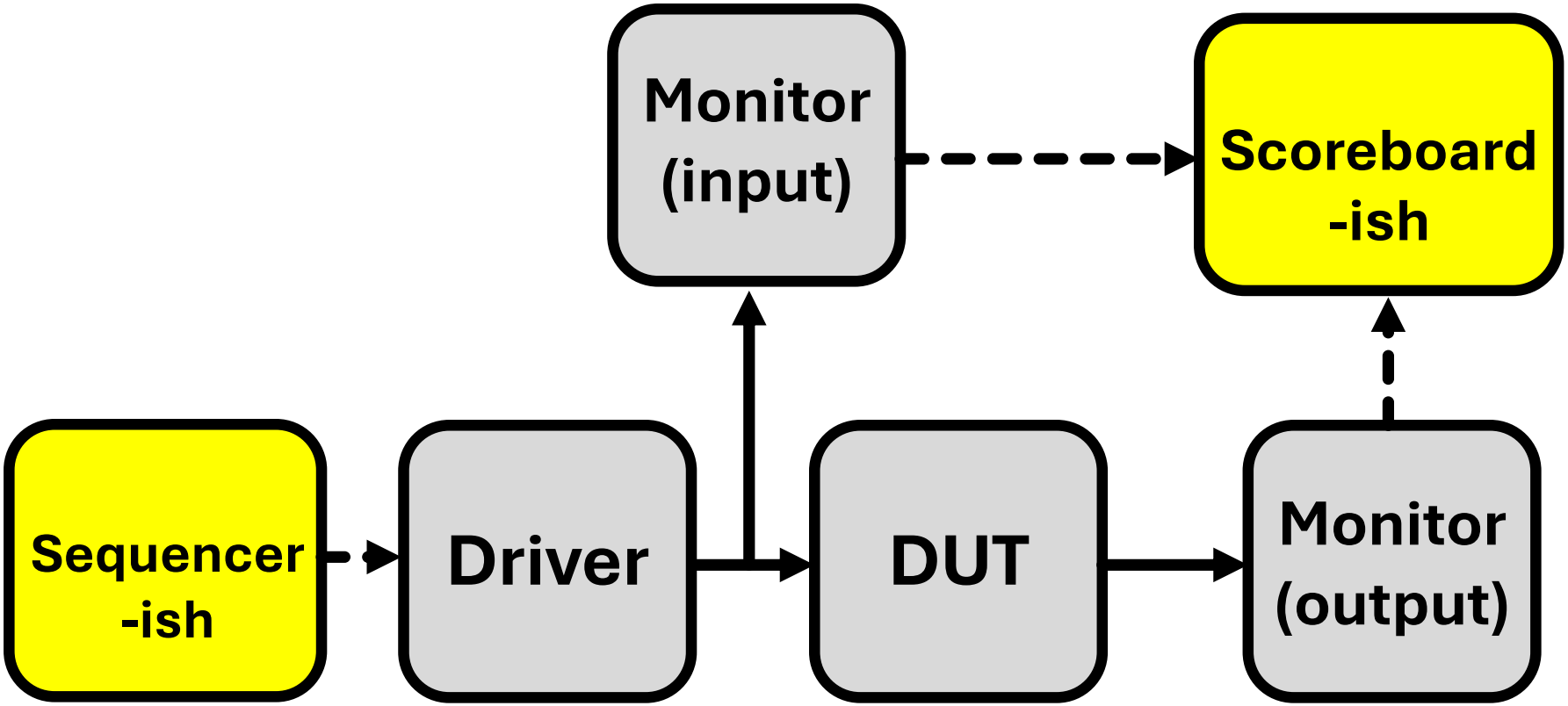
Used a "magic-eye" tube for tuning

Virginia Woolf's Radio at her house in Sussex, England

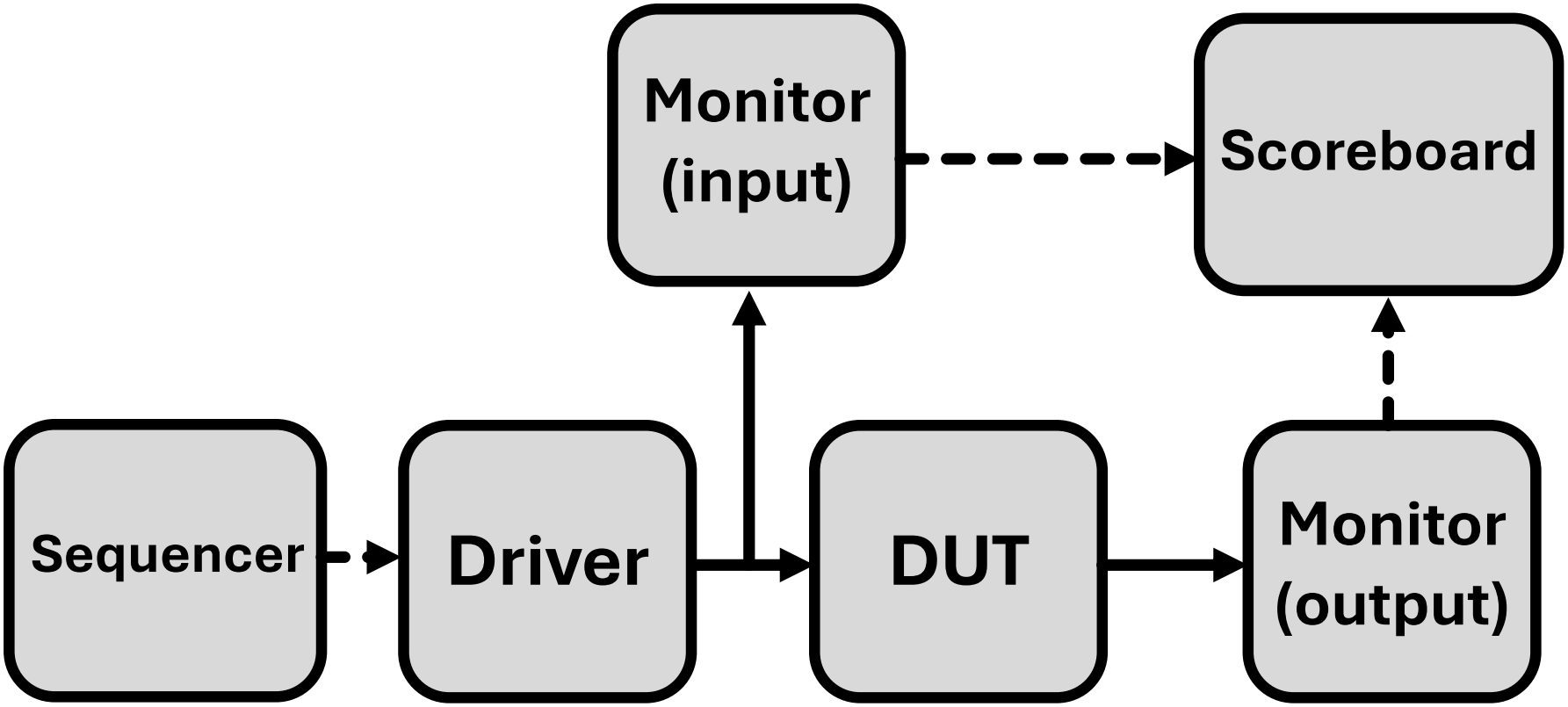
Standard Testing Framework



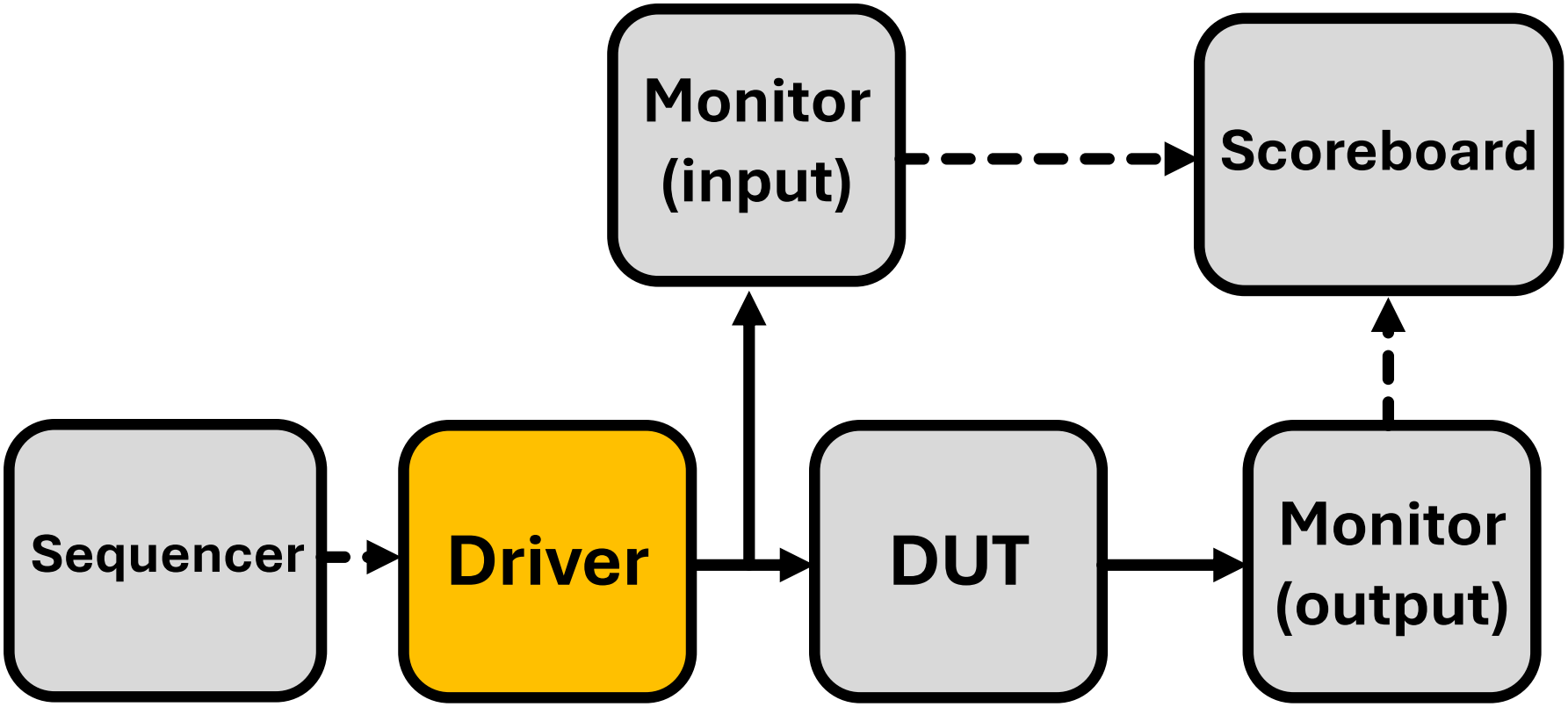
Week 3


Monitor
(input)
Scoreboard
-ish
Sequencer
-ish
Driver
DUT
Monitor
(output)

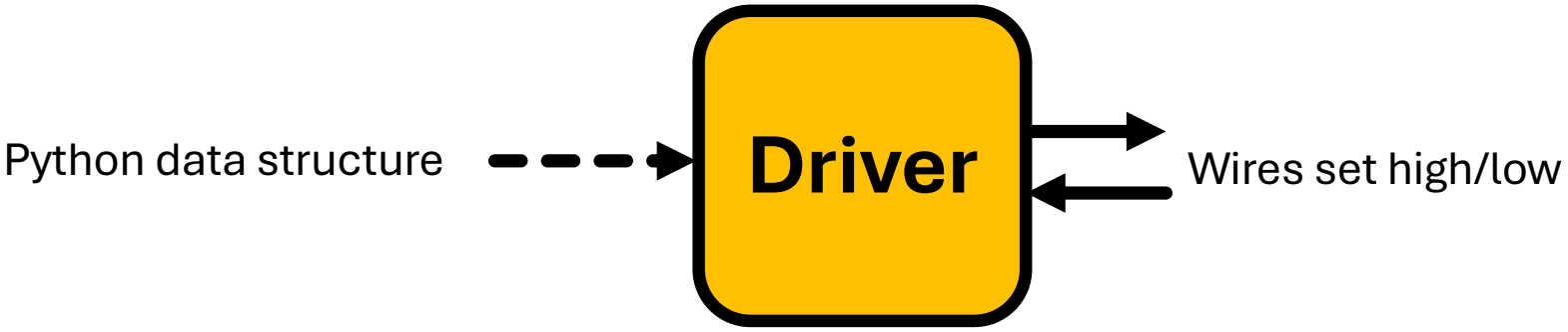
Standard Testing Framework



Standard Testing Framework



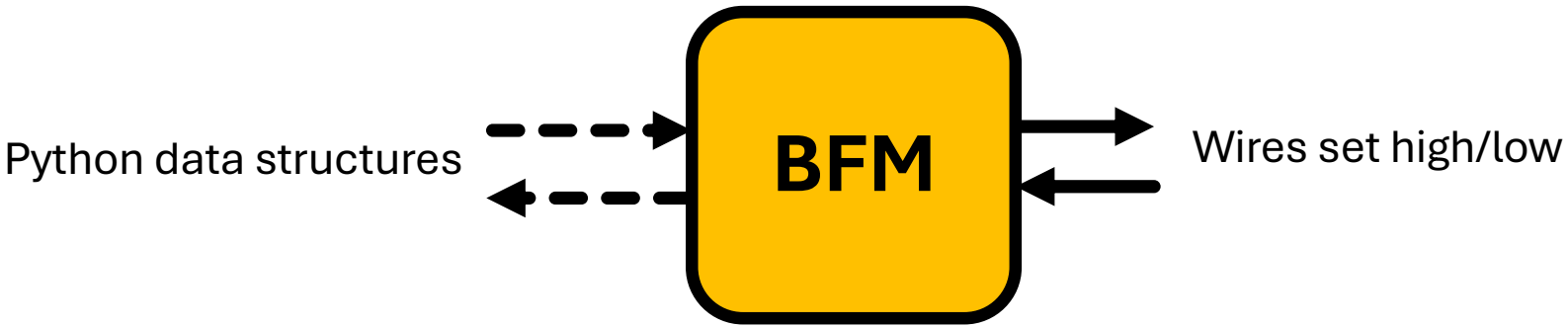
At a high level…



- Ideally, we want to just be able to issue high-level commands of things to send and not have to worry about:
  - Turning signals on/off
  - Waiting for signals to be on/off (e.g. READY)

Bus Functional Model: “BFM”

- You will find this term thrown around a lot and it is kinda/basically the same thing we’re doing here.
- A BFM is a programming construct that allows allows interfacing of testing languages/frameworks to work with the simulated digital designs



Drivers

- In week 3's stuff we're creating (and then using) a Bus Driver that you write like the following:

```
ind = AXISDriver(dut,'s00',dut.s00_axis_aclk)
#...
#...
#feed the driver:
for i in range(50):
  data = {'type':'single', "contents":{"data": random.randint(1,255),"last":0,"strb":15}}
  ind.append(data)
data = {'type':'burst', "contents":{"data": np.array(list(range(100)))}}
ind.append(data)
```

- What is this "append" method?

Drivers

- Several different classes and subclasses

cocotb-bus / src / cocotb_bus / drivers / __init__.py

```
21
22  class BitDriver:
23      """Drives a signal onto a single bit.
24
25      Useful for exercising ready/valid flags.
26      """
27
```

```
72
73  class Driver:
74      """Class defining the standard interface for a driver within a testbench.
75
76      The driver is responsible for serializing transactions onto the physical
77      pins of the interface.  This may consume simulation time.
78      """
```

```
205
206  class BusDriver(Driver):
207      """Wrapper around common functionality for buses which have:
208
209          * a list of :attr:`_signals` (class attribute)
210          * a list of :attr:`_optional_signals` (class attribute)
211          * a clock
212          * a name
```

```
286
287  class ValidatedBusDriver(BusDriver):
288      """Same as a :class:`BusDriver` except we support an optional generator
289      to control which cycles are valid.
290
```

The Driver Base Class

```
72
73  class Driver:
74      """Class defining the standard interface for a driver within a testbench.
75
76      The driver is responsible for serializing transactions onto the physical
77      pins of the interface.  This may consume simulation time.
78      """
79
80      def __init__(self):
81          """Constructor for a driver instance."""
82          self._pending = Event(name="Driver._pending")
83          self._sendQ = deque()
84          self.busy_event = Event("Driver._busy")
85          self.busy = False
86
87          # Sub-classes may already set up logging
88          if not hasattr(self, "log"):
89              self.log = logging.getLogger("cocotb.driver.%s" % (type(self).__qualname__))
90
91          # Create an independent coroutine which can send stuff
92          self._thread = cocotb.start_soon(self._send_thread())
93
94      async def _acquire_lock(self):
95          if self.busy:
96              await self.busy_event.wait()
97          self.busy_event.clear()
98          self.busy = True
99
```

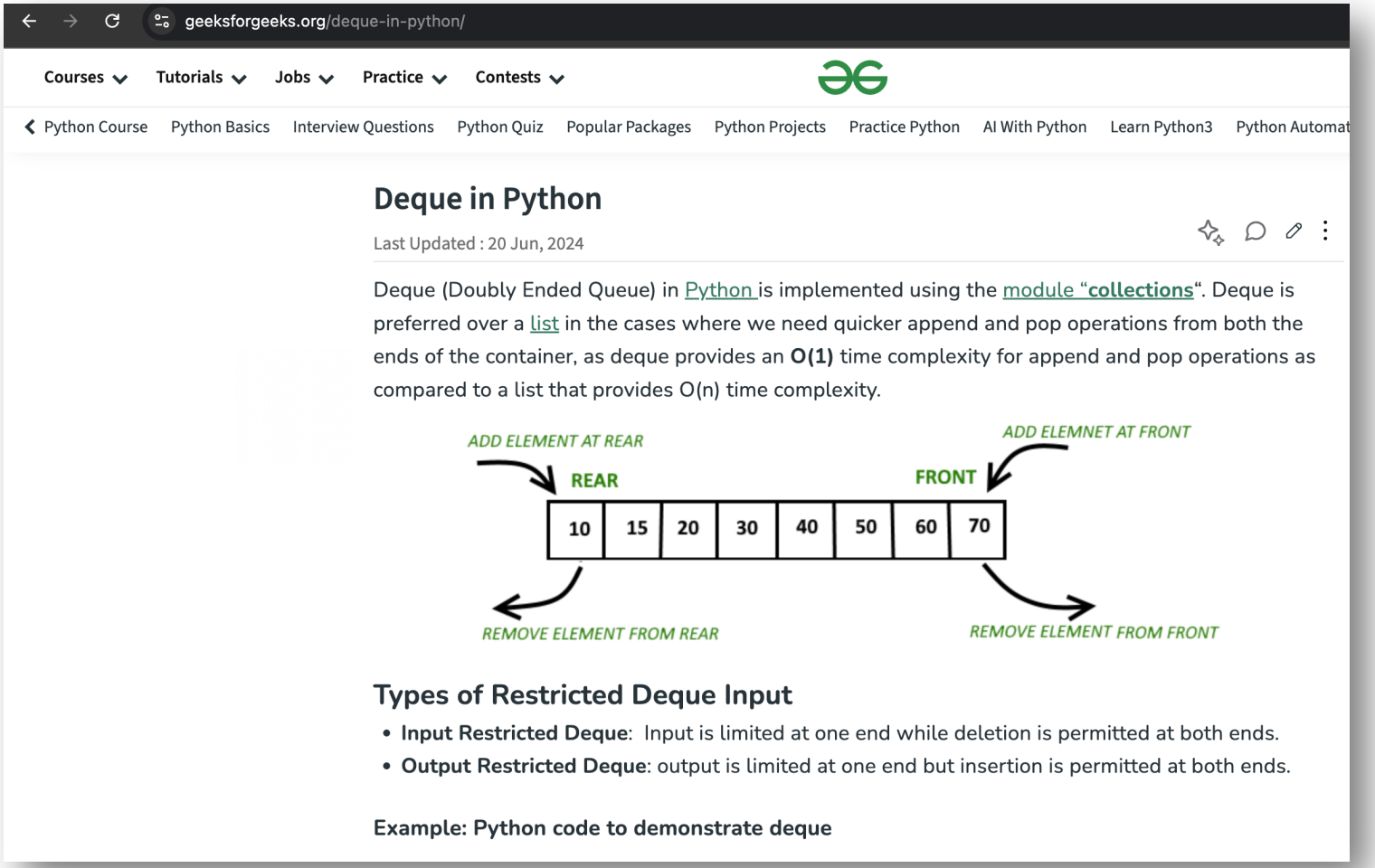
The append method

```
110      def append(
111          self, transaction: Any, callback: Callable[[Any], Any] = None,
112          event: Event = None, **kwargs: Any
113      ) -> None:
114          """Queue up a transaction to be sent over the bus.
115
116          Mechanisms are provided to permit the caller to know when the
117          transaction is processed.
118
119          Args:
120              transaction: The transaction to be sent.
121              callback: Optional function to be called
122                  when the transaction has been sent.
123              event: :class:`~cocotb.triggers.Event` to be set
124                  when the transaction has been sent.
125              **kwargs: Any additional arguments used in child class'
126                  :any:`_driver_send` method.
127          """
128          self._sendQ.append((transaction, callback, event, kwargs))
129          self._pending.set()
```

What is self._sendQ ?

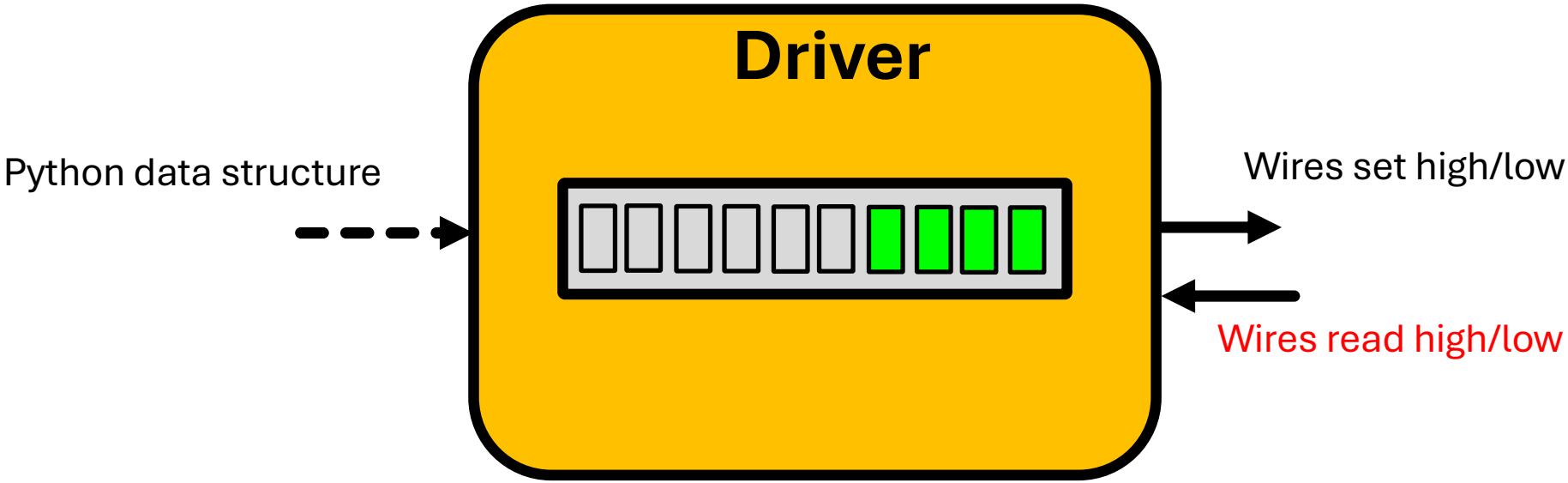
```
72
73  class Driver:
74      """Class defining the standard interface for a driver within a testbench.
75
76      The driver is responsible for serializing transactions onto the physical
77      pins of the interface.  This may consume simulation time.
78      """
79
80      def __init__(self):
81          """Constructor for a driver instance."""
82          self._pending = Event(name="Driver._pending")
83          self._sendQ = deque()
84          self.busy_event = Event("Driver._busy")
85          self.busy = False
86
87          # Sub-classes may already set up logging
88          if not hasattr(self, "log"):
89              self.log = logging.getLogger("cocotb.driver.%s" % (type(self).__qualname__))
90
91          # Create an independent coroutine which can send stuff
92          self._thread = cocotb.start_soon(self._send_thread())
93
```

Double Ended Queue



An internal FIFO of things to do

- Just like in hardware a FIFO/queue allows breathing room and a decoupling of commands from implementation

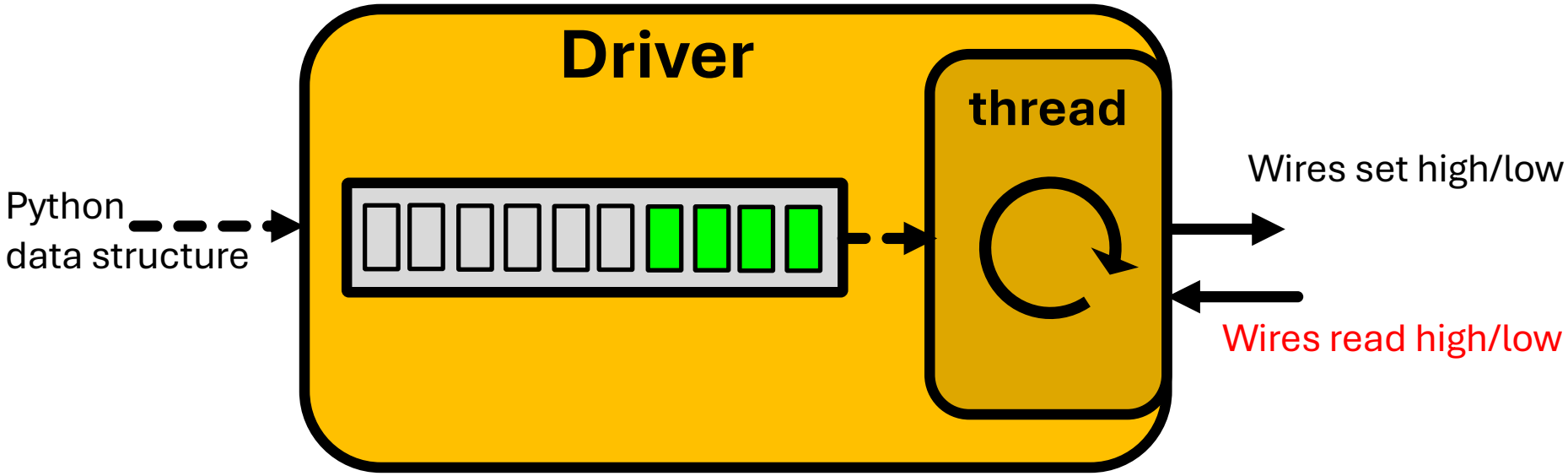


Launches the running process

```
72
73  class Driver:
74      """Class defining the standard interface for a driver within a testbench.
75
76      The driver is responsible for serializing transactions onto the physical
77      pins of the interface.  This may consume simulation time.
78      """
79
80      def __init__(self):
81          """Constructor for a driver instance."""
82          self._pending = Event(name="Driver._pending")
83          self._sendQ = deque()
84          self.busy_event = Event("Driver._busy")
85          self.busy = False
86
87          # Sub-classes may already set up logging
88          if not hasattr(self, "log"):
89              self.log = logging.getLogger("cocotb.driver.%s" % (type(self).__qualname__))
90
91          # Create an independent coroutine which can send stuff
92          self._thread = cocotb.start_soon(self._send_thread())
93
```

An internal FIFO of things to do

- Just like in hardware a FIFO/queue allows breathing room and a decoupling of commands from implementation



_send_thread coroutine

- Monitors queue…if stuff in it…calls the _send procedure

```
186      async def _send_thread(self):
187          while True:
188
189              # Sleep until we have something to send
190              while not self._sendQ:
191                  self._pending.clear()
192                  await self._pending.wait()
193
194              synchronised = False
195
196              # Send in all the queued packets,
197              # only synchronize on the first send
198              while self._sendQ:
199                  transaction, callback, event, kwargs = self._sendQ.popleft()
200                  self.log.debug("Sending queued packet...")
201                  await self._send(transaction, callback, event,
202                                   sync=not synchronised, **kwargs)
203                  synchronised = True
204
```

_send

- Finally this is getting to the _driver_send procedure which we have.

Notice this is awaiting it
And actually the last few pages have been awaits

```
163     async def _send(
164         self, transaction: Any, callback: Callable[[Any], Any], event: Event,
165         sync: bool = True, **kwargs
166     ) -> None:
167         """Send coroutine.
168
169         Args:
170             transaction: The transaction to be sent.
171             callback: Optional function to be called
172                 when the transaction has been sent.
173             event: event to be set when the transaction has been sent.
174             sync: Synchronize the transfer by waiting for a rising edge.
175             **kwargs: Any additional arguments used in child class'
176                 :any:`_driver_send` method.
177         """
178         await self._driver_send(transaction, sync=sync, **kwargs)
179
180         # Notify the world that this transaction is complete
181         if event:
182             event.set()
183         if callback:
184             callback(transaction)
```

What are we appending?

- Several Mechanisms for that.

```
110     def append(
111         self, transaction: Any, callback: Callable[[Any], Any] = None,
112         event: Event = None, **kwargs: Any
113     ) -> None:
114         """Queue up a transaction to be sent over the bus.
115
116         Mechanisms are provided to permit the caller to know when the
117         transaction is processed.
118
119         Args:
120             transaction: The transaction to be sent.
121             callback: Optional function to be called
122                 when the transaction has been sent.
123             event: :class:`~cocotb.triggers.Event` to be set
124                 when the transaction has been sent.
125             **kwargs: Any additional arguments used in child class'
126                 :any:`_driver_send` method.
127         """
128         self._sendQ.append((transaction, callback, event, kwargs))
129         self._pending.set()
```

We may also want to get feedback from our Drivers

- Several Mechanisms for that.

```
110     def append(
111         self, transaction: Any, callback: Callable[[Any], Any] = None,
112         event: Event = None, **kwargs: Any
113     ) -> None:
114         """Queue up a transaction to be sent over the bus.
115
116         Mechanisms are provided to permit the caller to know when the
117         transaction is processed.
118
119         Args:
120             transaction: The transaction to be sent.
121             callback: Optional function to be called
122                 when the transaction has been sent.
123             event: :class:`~cocotb.triggers.Event` to be set
124                 when the transaction has been sent.
125             **kwargs: Any additional arguments used in child class'
126                 :any:`_driver_send` method.
127         """
128         self._sendQ.append((transaction, callback, event, kwargs))
129         self._pending.set()
```

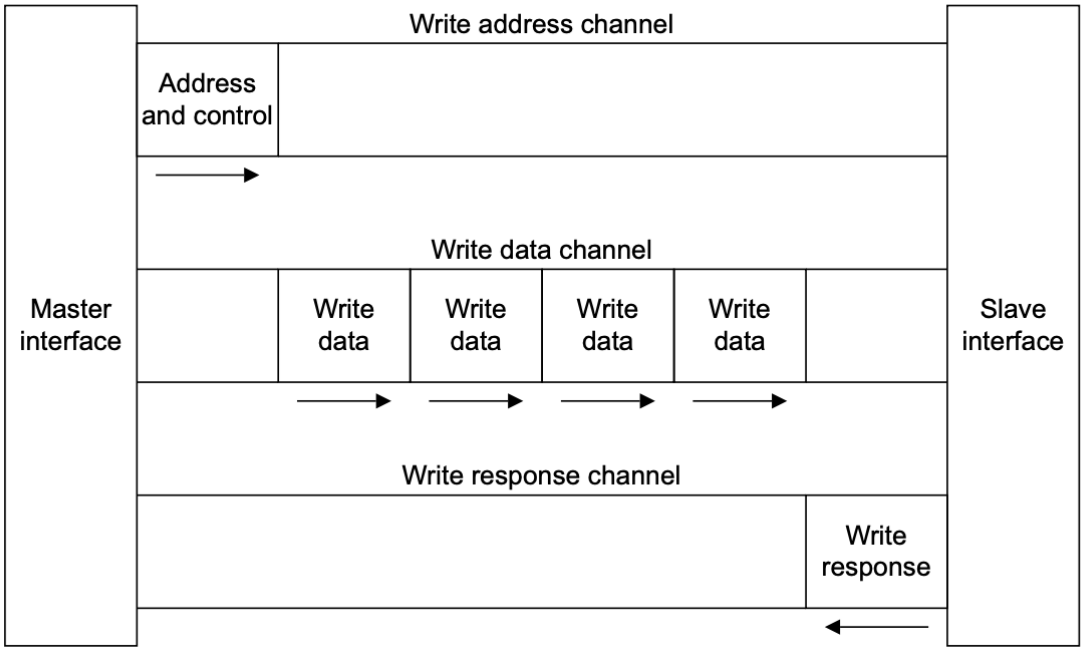
Callback?

- A callback is essentially a function that gets called when an transaction is complete.
- You could:
  - Log the event that you wrote it
  - Print something
  - Perhaps call some other function that updates global state to influence future behavior (useful for synchronizing multiple drivers)

Event?

- These are a second way of causing a python-side thing to occur when a transaction happens
- These are a fundamental part of the cocotb library, allowing different coroutines to communicate with each other

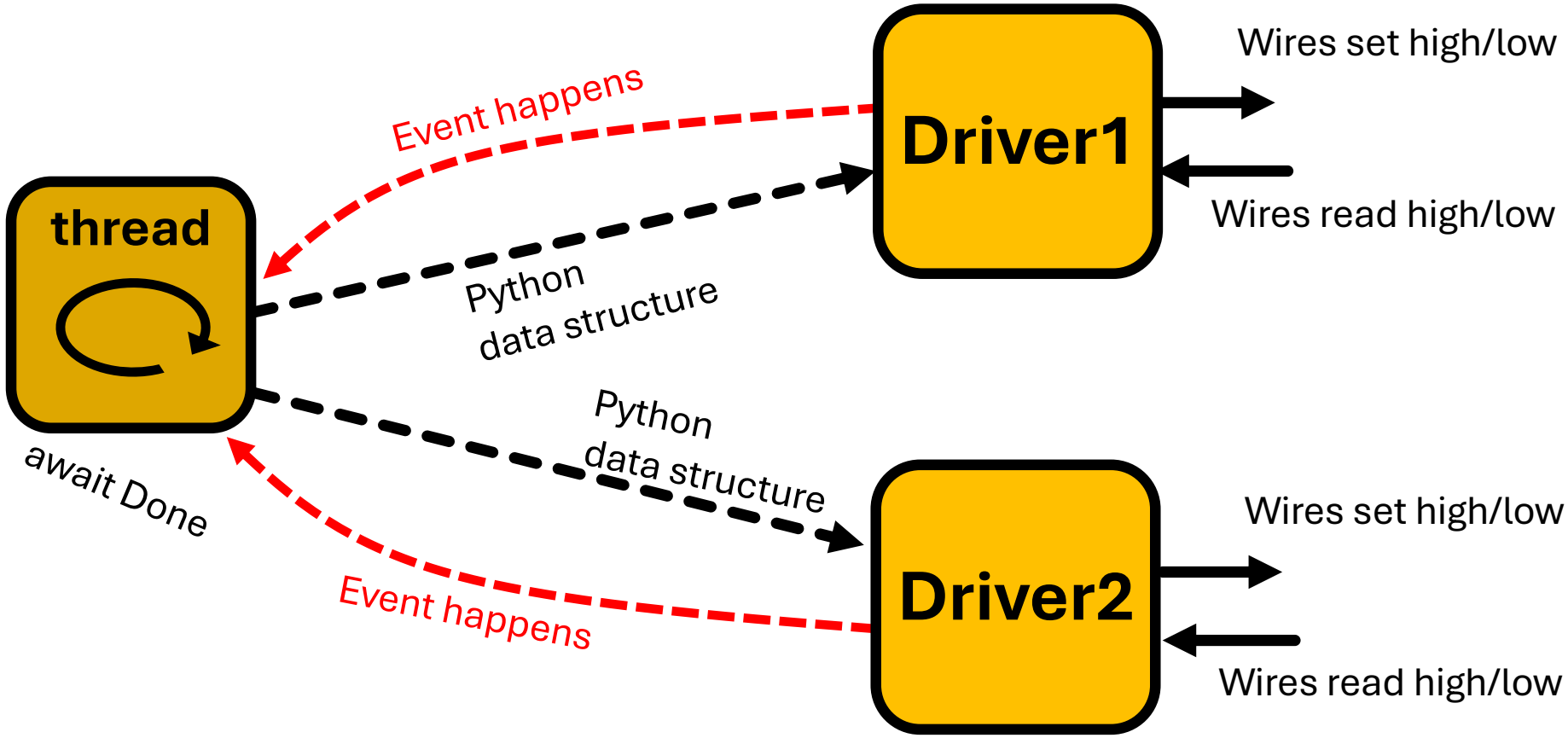
Usefulness? Think of the AXI LITE



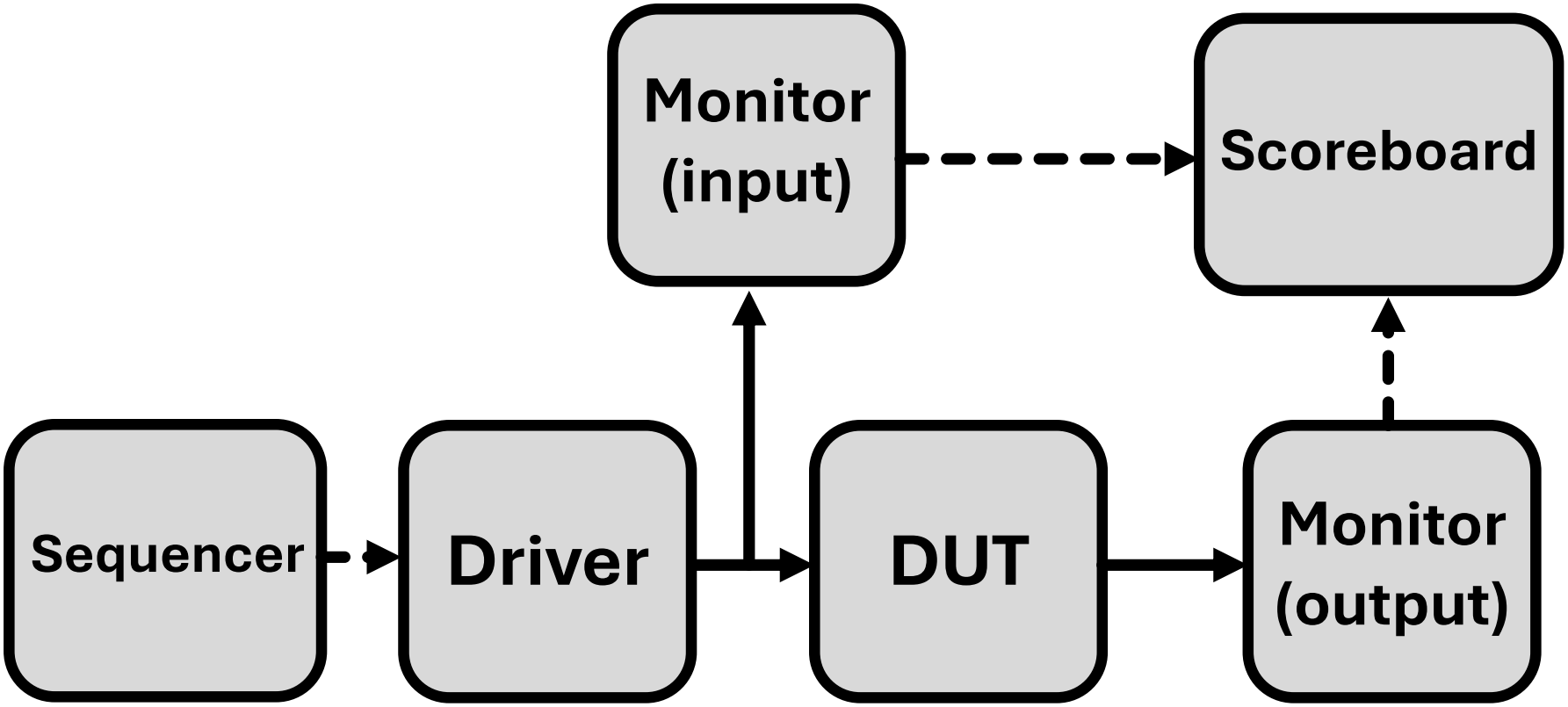
- One interface actually has three separate busses in it. Have Driver for Each Bus, but need to sync them…

Events

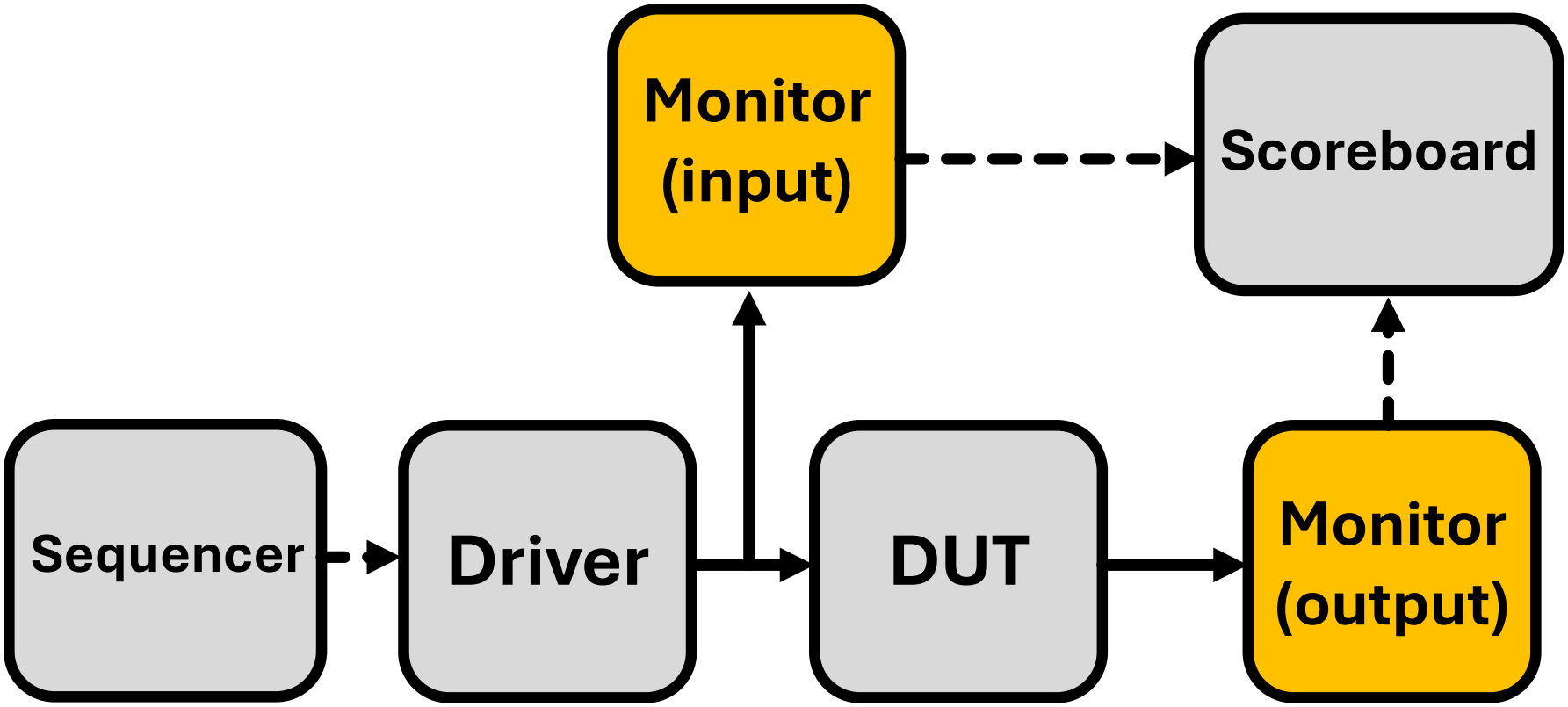
- Events/Triggers could allow Driver 1 to only send after Driver 2 sent or vice versa or whatever



Standard Testing Framework



Standard Testing Framework

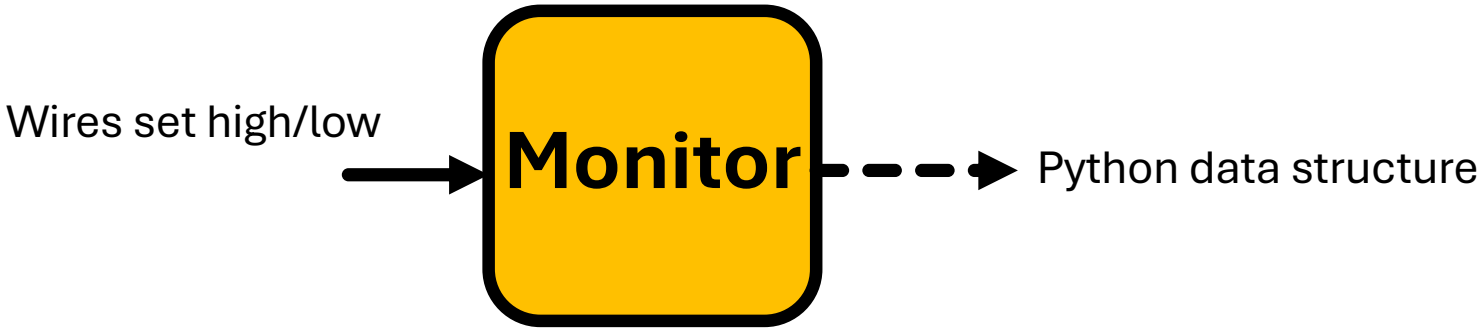


AXISMonitor

```python
class AXISMonitor(BusMonitor):
    """
    monitors axi streaming bus
    """
    transactions = 0
    def __init__(self, dut, name, clk):
        self._signals = ['axis_tvalid','axis_tready','axis_tlast','axis_tdata','axis_tstrb']
        BusMonitor.__init__(self, dut, name, clk)
        self.clock = clk
        self.transactions = 0
    async def _monitor_recv(self):
        """
        Monitor receiver
        """
        rising_edge = RisingEdge(self.clock) # make these coroutines once and reuse
        falling_edge = FallingEdge(self.clock)
        read_only = ReadOnly() #This is
        while True:
            await rising_edge
            await falling_edge #sometimes see in AXI shit
            await read_only  #readonly (the postline)
            valid = self.bus.axis_tvalid.value
            ready = self.bus.axis_tready.value
            last = self.bus.axis_tlast.value
            data = self.bus.axis_tdata.value #.signed_integer
            if valid and ready:
              self.transactions+=1
              thing = dict(data=data,last=last,name=self.name,count=self.transactions,time=gst())
              print(thing)
              self._recv(thing)
```

- Given to you in week 3
- Monitors the line
- If a valid/ready transaction occurs, do something with it.

At a high level…



- Monitors should be completely passive entities that simply report what they see and not affect the system
- Keeps it simple

A little closer

```
if valid and ready:
    self.transactions+=1
    thing = dict(data=data,last=last,name=self.name,count=self.transactions,time=gst())
    print(thing)
    self._recv(thing)
```

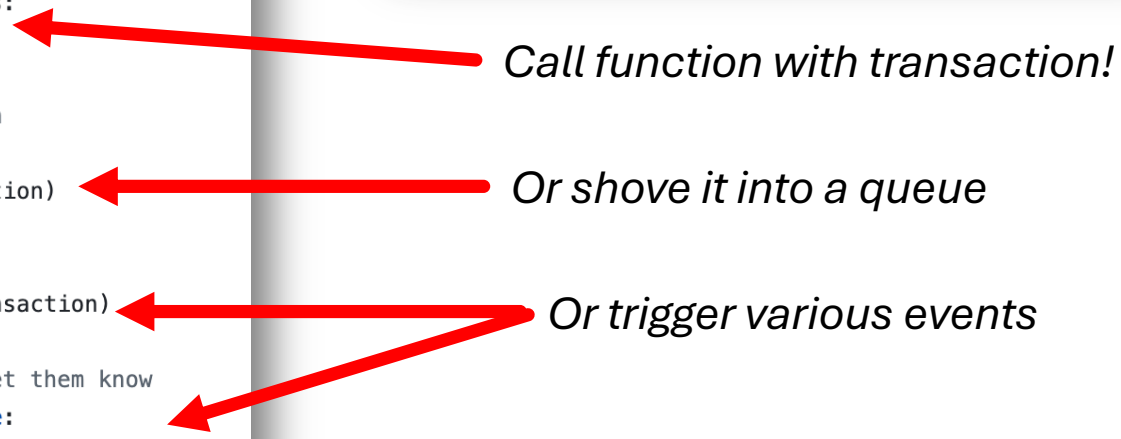
- Made a internal variable to keep track of the number of valid/ready things we saw happen on the bus
- Then created a Python data structure from it (for easy human interpretability) and reported it.

The _recv method does a few things:

__init__ of Monitor Class:

```
def __init__(self, callback=None, event=None):
    self._event = event
    if self._event is not None:
        self._event.data = None  # FIXME: This
    self._wait_event = Event()
    self._wait_event.data = None
    self._recvQ = deque()
    self._callbacks = []
```

```
126     def _recv(self, transaction):
127         """Common handling of a received transaction."""
128
129         self.stats.received_transactions += 1
130
131         # either callback based consumer
132         for callback in self._callbacks:
133             callback(transaction)
134
135         # Or queued with a notification
136         if not self._callbacks:
137             self._recvQ.append(transaction)
138
139         if self._event is not None:
140             set_event(self._event, transaction)
141
142         # If anyone was waiting then let them know
143         if self._wait_event is not None:
144             set_event(self._wait_event, transaction)
145             self._wait_event.clear()
```



Monitor Class

```
29
30  class Monitor:
31      """Base class for Monitor objects.
32
33      Monitors are passive 'listening' objects that monitor pins going in or out of a DUT.
34      This class should not be used directly,
35      but should be sub-classed and the internal :meth:`_monitor_recv` method should be overridde
36      This :meth:`_monitor_recv` method should capture some behavior of the pins, form a transact
37      and pass this transaction to the internal :meth:`_recv` method.
38      The :meth:`_monitor_recv` method is added to the cocotb scheduler during the ``__init__`` p
39      so it should not be awaited anywhere.
40
41      The primary use of a Monitor is as an interface for a :class:`~cocotb.scoreboard.Scoreboard
42
43      Args:
44          callback (callable): Callback to be called with each recovered transaction
45              as the argument. If the callback isn't used, received transactions will
46              be placed on a queue and the event used to notify any consumers.
47          event (cocotb.triggers.Event): Event that will be called when a transaction
48              is received through the internal :meth:`_recv` method.
49              `Event.data` is set to the received transaction.
50      """
51
```

What would the monitor/callbacks be useful for?

- Right now (week 3), just printing stuff and verify by eye

```
{'data': 00000000000000000010100000110011, 'last': 0, 'name': 'm00', 'count': 150, 'time': 5105000}
{'data': 00000000000000000000000001100011, 'last': 1, 'name': 's00', 'count': 150, 'time': 5115000}
{'data': 00000000000000000010100000110110, 'last': 0, 'name': 'm00', 'count': 151, 'time': 5115000}
{'data': 00000000000000000010100000111001, 'last': 1, 'name': 'm00', 'count': 152, 'time': 5125000}
```

- But it would be nice to verify the data, for example… That's where maybe a callback could come in

One useful callback might be a model

- You wrote a model in week 1 when “verifying” that crappy divider we gave you
- A callback to a model might be useful if attached to an input monitor.
- Every time an input to DUT is observed, you trigger the model to compute what to expect off of it.

Minor modifications to BusMonitor

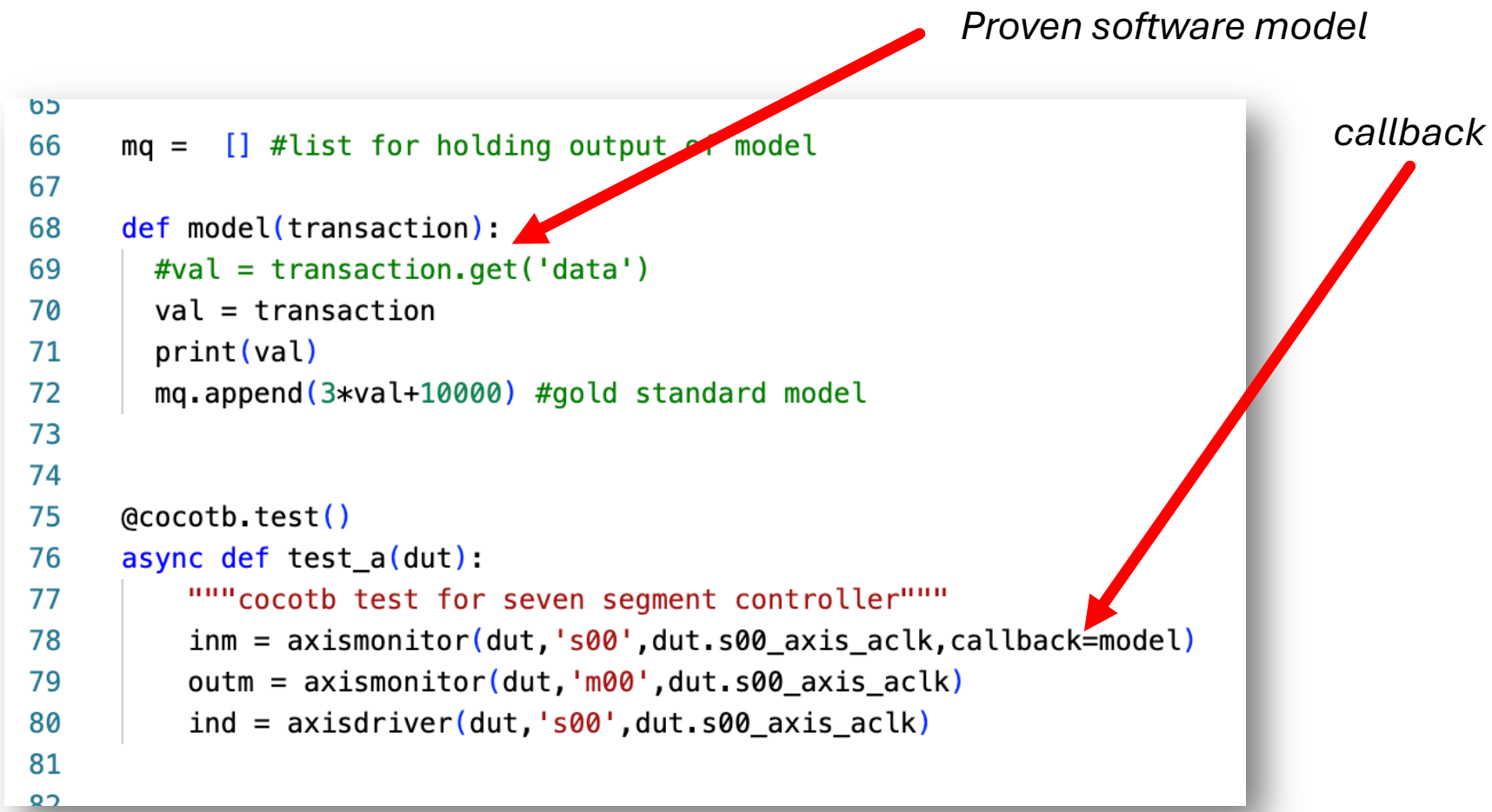
callback

```
52
53  class AXISMonitor(BusMonitor):
54      """
55      monitors axi streaming bus
56      """
57      transactions = 0
58      def __init__(self, dut, name, clk, callback=None):
59          self._signals = ['axis_tvalid','axis_tready','axis_tlast','axis_tdata','axis_tstrb']
60          BusMonitor.__init__(self, dut, name, clk, callback=callback)
61          self.clock = clk
62          self.transactions = 0
63
```

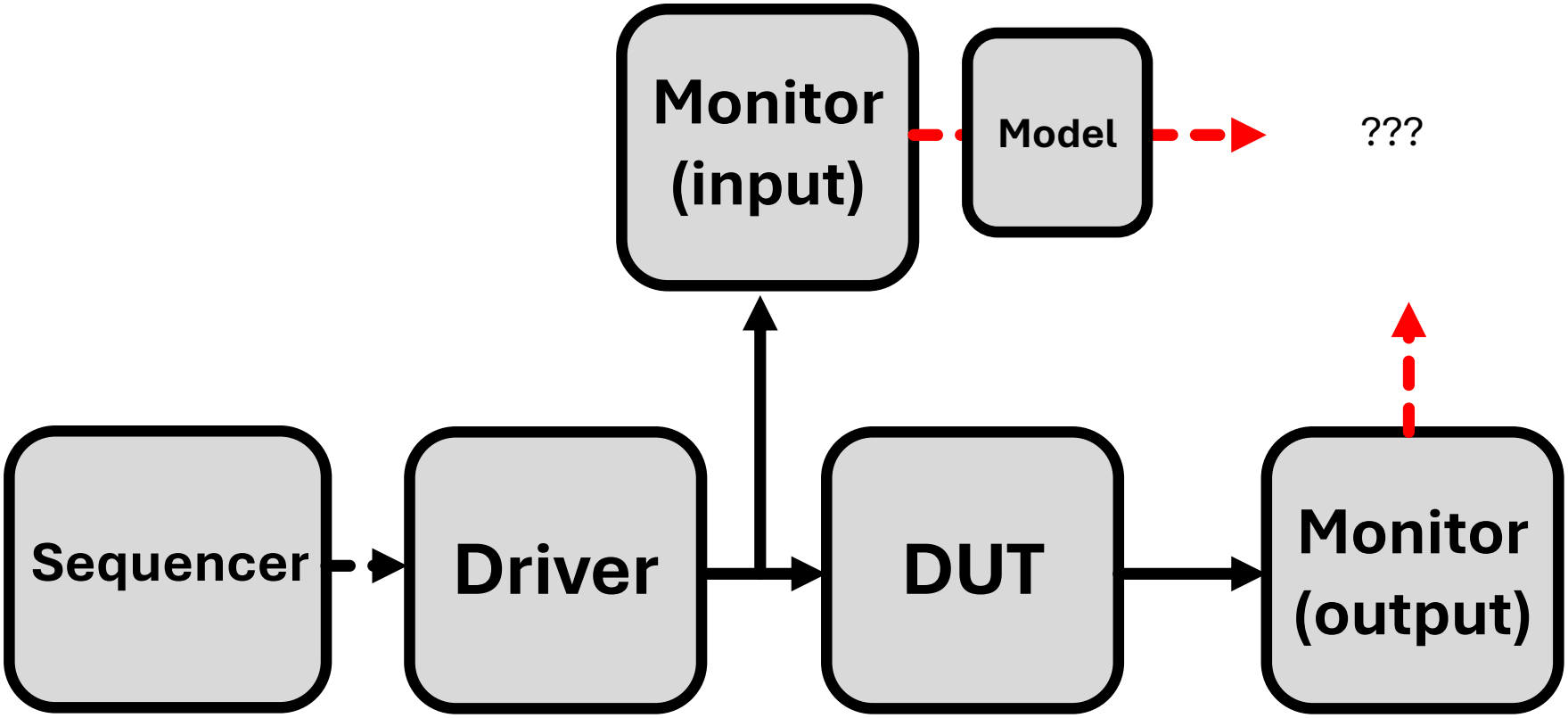
```
if valid and ready:
  self.transactions+=1
  thing = dict(data=data,last=last,name=self.name,count=self.transactions,time=gst())
  print(thing)
  self._recv(data.value)
```

Just send the value to that thing rather than dictionary thing

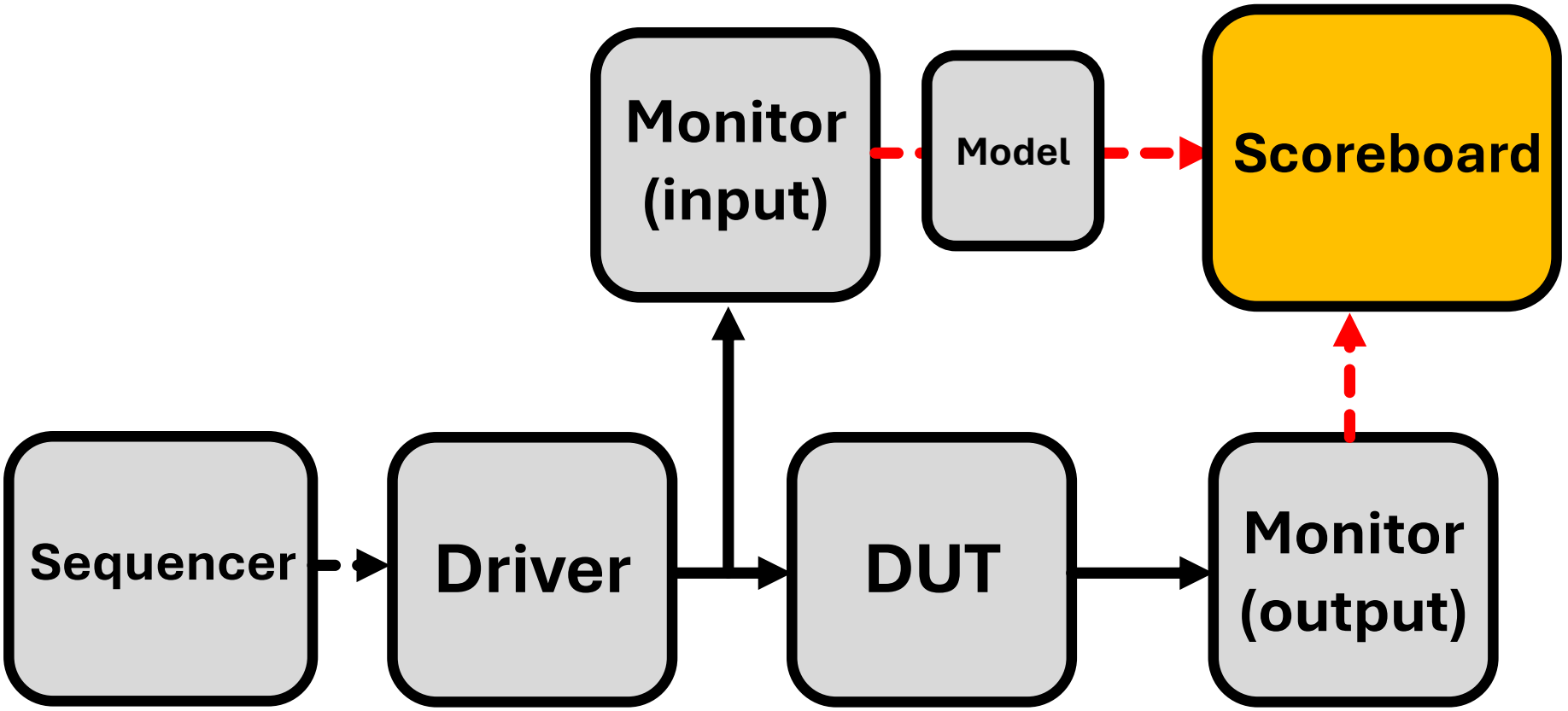
With these modifications…



So now



Plug an entity in to Compare these results



ScoreBoard

- There's a scoreboarding class that is designed to work with data streams like this

```
class Scoreboard:
    """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.

    Args:
        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 :exc:`AssertionError`
            immediately when something is wrong instead of just
            recording an error. Default is ``True``.
    """

    def __init__(self, dut, reorder_depth=0, fail_immediately=True):  # FIXME: reorder_depth ne
        self.dut = dut
        self.log = logging.getLogger("cocotb.scoreboard.%s" % self.dut._name)
        self.errors = 0
        self.expected = {}
        self._imm = fail_immediately

    @property
    def result(self):
        """Determine the test result, do we have any pending data remaining?

        Raises:
```

Make a scoreboard

```
@cocotb.test()
async def test_a(dut):
    """cocotb test for seven segment controller"""
    inm = AXISMonitor(dut,'s00',dut.s00_axis_aclk,callback=model)
    outm = AXISMonitor(dut,'m00',dut.s00_axis_aclk)
    ind = AXISDriver(dut,'s00',dut.s00_axis_aclk)
    scoreboard = Scoreboard(dut)
    scoreboard.add_interface(outm,mq)
```

Scoreboard instance

Thing for it to check…

actual, expected

Scoreboard Class

- Has all the stuff running to check/compare the actual/expected pairs as they come in.
- Can also override the compare to do whatever you want…ranges, whatever

```
def compare(self, got, exp, log, strict_type=True):
    """Common function for comparing two transactions.

    Can be re-implemented by a sub-class.

    Args:
        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:
        :exc:`AssertionError`: If received transaction differed from
            expected transaction when :attr:`fail_immediately` is ``True``.
            If *strict_type* is ``True``,
            also the transaction type must match.
    """
```

Scoreboard sees failure and tells you

```
/Users/jodalyst/cocotb_development/fir_dev2/sim/test_fir.py:118: DeprecationWarning: Use `bv.integer` instead.
  self._recv(data.value)
    35.00ns ERROR    cocotb.scoreboard.j_math.m00       Received transaction differed from expected output
    35.00ns INFO     cocotb.scoreboard.j_math.m00       Expected:
                                                        20312
    35.00ns INFO     cocotb.scoreboard.j_math.m00       Received:
                                                        10312
/Users/jodalyst/6205_python/lib/python3.10/site-packages/cocotb_bus/scoreboard.py:140: DeprecationWarning: cocotb.utils.hexdiffs is deprecated. Use scapy.utils.hexdiff instead.
  log.warning("Difference:\n%s" % hexdiffs(strexp, strgot))
/Users/jodalyst/6205_python/lib/python3.10/site-packages/cocotb_bus/scoreboard.py:140: DeprecationWarning: Passing strings to hexdiffs is deprecated, pass bytes instead
  log.warning("Difference:\n%s" % hexdiffs(strexp, strgot))
    35.00ns WARNING  cocotb.scoreboard.j_math.m00       Difference:
                                                        0000      3230333132                            20312
                                                             0000 3130333132                            10312

/Users/jodalyst/6205_python/lib/python3.10/site-packages/cocotb_bus/scoreboard.py:142: DeprecationWarning: TestFailure is deprecated, use an ``assert`` statement instead
  raise TestFailure("Received transaction differed from expected "
    35.00ns INFO     ..Task 1.AXISMonitor._monitor_recv Test stopped by this forked coroutine
    35.00ns INFO     cocotb.regression                  test_a failed
                                                        Traceback (most recent call last):
                                                          File "/Users/jodalyst/cocotb_development/fir_dev2/sim/test_fir.py", line 118, in _monitor_recv
                                                            self._recv(data.value)
                                                          File "/Users/jodalyst/6205_python/lib/python3.10/site-packages/cocotb_bus/monitors/__init__.py", line 130, in _recv
                                                            callback(transaction)
                                                          File "/Users/jodalyst/6205_python/lib/python3.10/site-packages/cocotb_bus/scoreboard.py", line 227, in check_received_tran
saction
                                                            self.compare(transaction, exp, log, strict_type=strict_type)
                                                          File "/Users/jodalyst/6205_python/lib/python3.10/site-packages/cocotb_bus/scoreboard.py", line 142, in compare
                                                            raise TestFailure("Received transaction differed from expected "
                                                        cocotb.result.TestFailure: Received transaction differed from expected transaction
    35.00ns INFO     cocotb.regression                  ****************************************************************************************
                                                        ** TEST                          STATUS  SIM TIME (ns)  REAL TIME (s)  RATIO (ns/s) **
                                                        ****************************************************************************************
                                                        ** test_fir.test_a                FAIL          35.00           0.00      12019.39  **
                                                        ****************************************************************************************
                                                        ** TESTS=1 PASS=0 FAIL=1 SKIP=0                35.00           0.04        843.29  **
                                                        ****************************************************************************************
```

Scoreboard see no error and you’re good

```
                                        ** test_fir.test_a               FAIL          35.00           0.00      12019.39  **
                                        ***************************************************************************************
                                        ** TESTS=1 PASS=0 FAIL=1 SKIP=0                35.00           0.04        843.29  **
                                        ***************************************************************************************

INFO: Results file: /Users/jodalyst/cocotb_development/fir_dev2/sim/sim_build/results.xml
(6205_python) (base) DHCP-POOL-18-25-22-252:sim jodalyst$ python3 test_fir.py
/Users/jodalyst/cocotb_development/fir_dev2/sim/test_fir.py:12: UserWarning: Python runners and associated APIs are an experimental feature and subject to change.
  from cocotb.runner import get_runner
INFO: Running command iverilog -o /Users/jodalyst/cocotb_development/fir_dev2/sim/sim_build/sim.vvp -D COCOTB_SIM=1 -s j_math -g2012 -Wall -s cocotb_iverilog_dump -f /Users/jodalyst/
cocotb_development/fir_dev2/sim/sim_build/cmds.f /Users/jodalyst/cocotb_development/fir_dev2/hdl/j_math.sv /Users/jodalyst/cocotb_development/fir_dev2/sim/sim_build/cocotb_iverilog_d
ump.v in directory /Users/jodalyst/cocotb_development/fir_dev2/sim/sim_build
INFO: Running command vvp -M /Users/jodalyst/6205_python/lib/python3.10/site-packages/cocotb/libs -m libcocotbvpi_icarus /Users/jodalyst/cocotb_development/fir_dev2/sim/sim_build/sim
.vvp in directory /Users/jodalyst/cocotb_development/fir_dev2/sim/sim_build
     -.--ns INFO     gpi                                ..mbed/gpi_embed.cpp:109  in set_program_name_in_venv       Using Python virtual environment interpreter at /Users/jodalyst/6
205_python/bin/python
     -.--ns INFO     gpi                                ../gpi/GpiCommon.cpp:101  in gpi_print_registered_impl      VPI registered
     0.00ns INFO     cocotb                             Running on Icarus Verilog version 12.0 (stable)
     0.00ns INFO     cocotb                             Running tests with cocotb v1.9.1 from /Users/jodalyst/6205_python/lib/python3.10/site-packages/cocotb
     0.00ns INFO     cocotb                             Seeding Python random module with 1727293312
     0.00ns INFO     cocotb.regression                  pytest not found, install it to enable better AssertionError messages
/Users/jodalyst/cocotb_development/fir_dev2/sim/test_fir.py:12: UserWarning: Python runners and associated APIs are an experimental feature and subject to change.
  from cocotb.runner import get_runner
     0.00ns INFO     cocotb.regression                  Found test test_fir.test_a
     0.00ns INFO     cocotb.regression                  running test_a (1/1)
                                                          cocotb test for seven segment controller
/Users/jodalyst/6205_python/lib/python3.10/site-packages/cocotb_bus/monitors/__init__.py:67: DeprecationWarning: This method is now private.
  self._thread = cocotb.scheduler.add(self._monitor_recv())
/Users/jodalyst/6205_python/lib/python3.10/site-packages/cocotb_bus/drivers/__init__.py:92: DeprecationWarning: This method is now private.
  self._thread = cocotb.scheduler.add(self._send_thread())
     0.00ns INFO     cocotb.scoreboard.j_math           Created with reorder_depth 0
VCD info: dumpfile /Users/jodalyst/cocotb_development/fir_dev2/sim/sim_build/j_math.fst opened for output.
/Users/jodalyst/cocotb_development/fir_dev2/sim/test_fir.py:170: DeprecationWarning: Setting values on handles using the ``dut.handle = value`` syntax is deprecated. Instead use the
``handle.value = value`` syntax
  dut.m00_axis_tready = val
/Users/jodalyst/cocotb_development/fir_dev2/sim/test_fir.py:118: DeprecationWarning: Use `bv.integer` instead.
  self._recv(data.value)
  6720.00ns INFO     cocotb.regression                  test_a passed
  6720.00ns INFO     cocotb.regression                  ***************************************************************************************
                                        ** TEST                          STATUS  SIM TIME (ns)  REAL TIME (s)  RATIO (ns/s) **
                                        ***************************************************************************************
                                        ** test_fir.test_a                PASS        6720.00           0.05     129322.58  **
                                        ***************************************************************************************
                                        ** TESTS=1 PASS=1 FAIL=0 SKIP=0              6720.00           0.09      72405.14  **
                                        ***************************************************************************************

INFO: Results file: /Users/jodalyst/cocotb_development/fir_dev2/sim/sim_build/results.xml
(6205_python) (base) DHCP-POOL-18-25-22-252:sim jodalyst$
```

Places For Improvement at end of Week 3

```python
@cocotb.test()
async def test_a(dut):
    """cocotb test for seven segment controller"""
    inm = AXISMonitor(dut,'s00',dut.s00_axis_aclk,callback=model)
    outm = AXISMonitor(dut,'m00',dut.s00_axis_aclk)
    ind = AXISDriver(dut,'s00',dut.s00_axis_aclk)
    scoreboard = Scoreboard(dut)
    scoreboard.add_interface(outm,mq)
    cocotb.start_soon(Clock(dut.s00_axis_aclk, 10, units="ns").start())
    await set_ready(dut,1)
    await reset(dut.s00_axis_aclk, dut.s00_axis_aresetn,2,0)
    #feed the driver:
    for i in range(50):
      data = {'type':'single', "contents":{"data": random.randint(1,255),"last":0,"strb":15}}
      ind.append(data)
    #data = {'type':'burst', "contents":{"data": np.array(20*[0]+[1]+30*[0]+[-2]+59*[0])}}
    data = {'type':'burst', "contents":{"data": np.array(list(range(100)))}}
    ind.append(data)
    await ClockCycles(dut.s00_axis_aclk, 50)
    await set_ready(dut,0)
    await ClockCycles(dut.s00_axis_aclk, 300)
    await set_ready(dut,1)
    await ClockCycles(dut.s00_axis_aclk, 10)
    await set_ready(dut,0)
    await ClockCycles(dut.s00_axis_aclk, 10)
    await set_ready(dut,1)
    await ClockCycles(dut.s00_axis_aclk, 300)
    assert inm.transactions==outm.transactions, f"Transaction Count doesn't match! :/"
```

Our creation of stuff is getting better but it would still be nicer for that to be more high-level…

Still a little ugly

Maybe make driver for output port

Sequencer will fix some of that