6.S965 Digital Systems Laboratory II

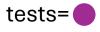
Lecture 11

Administrative Stuff

- Last week of stuff (week 6) content was released Friday
- Try to do by the end of the week.
- No idea why the RFSoC samples at half the specified rate.
 - Anybody figure that out they'll get three US Dollars
- I'm going through projects and things now. Sorry 6.205 delayed me. Feel free to also reach out/post ideas on Piazza if you are looking for a team

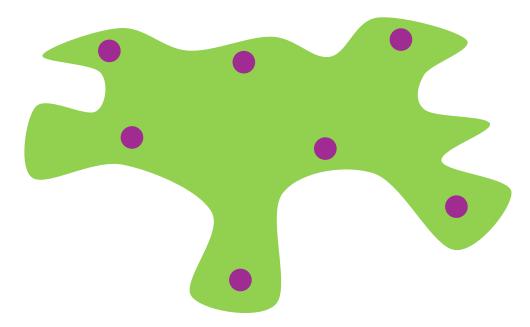
Coverage

What is it? What does it mean?



Coverage

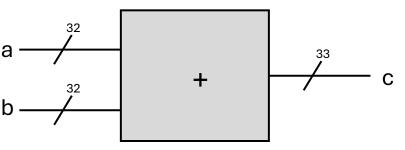
 Is concerned with how much have you tested a DUT



Some n-dimensional blob of possible module existence

The issue...

• Consider a device that adds two 32 bit numbers.



- There are 1.84×10^{19} input possibilities, each with a correct output.
- If you verified 1 billion input/output combinations per second it would take ~600 years to fully verify the design
- And this is just a simple adder...

And this gets astronomically worse as modules get more complicated

- ...especially as they get more stateful
- ...and with more inputs
- ...and with multiple sets of ports and things

Can anything ever be fully covered?

- Some modules should be able to be almost fully covered
- Others maybe not, so you have to structure what you're looking for and zero in on important edge cases like:
 - Max/min values, edge cases, overflow cases,

What do you "cover"?

- If a module has clearly defined states, you should check to see those
- Maybe check to see how those states transition?
- Maybe check to see different sequences of input and/or output signals
- Check certain output signals against input signals
- Check sequences of inputs

Coverage is not necessarily about the verification of correct results

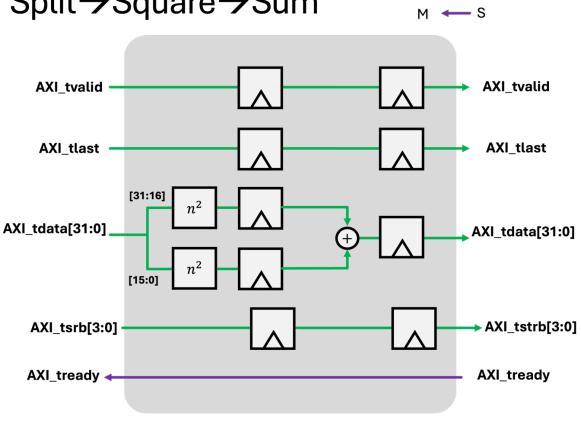
- I mean it is an adjacent topic
- But really the notion of coverage is meant to say *how* much was tested...with the assumption that it tested correctly.
- It is also about exploring what/where your design can get to and can't get to.

So let's look at an example...

- We'll revisit the issue of TREADY propagation and build a module to handle that properly.
- This plagued some people in earlier weeks.

Week 4 Split-Square-Sum

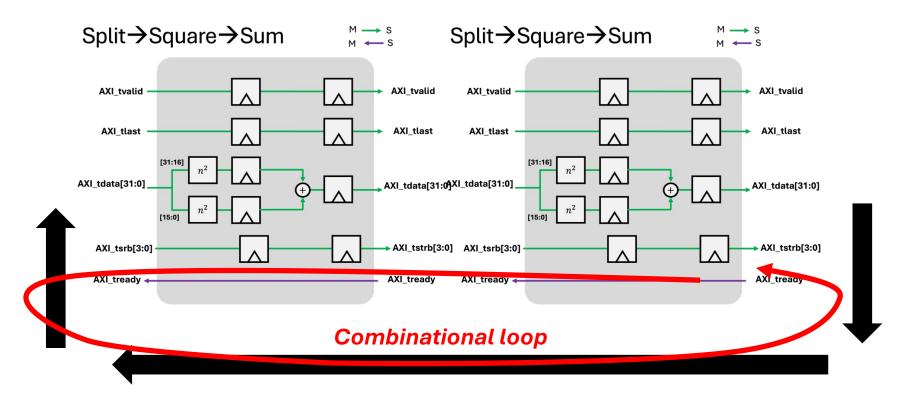
• Any Problems? Split→Square→Sum



M — S

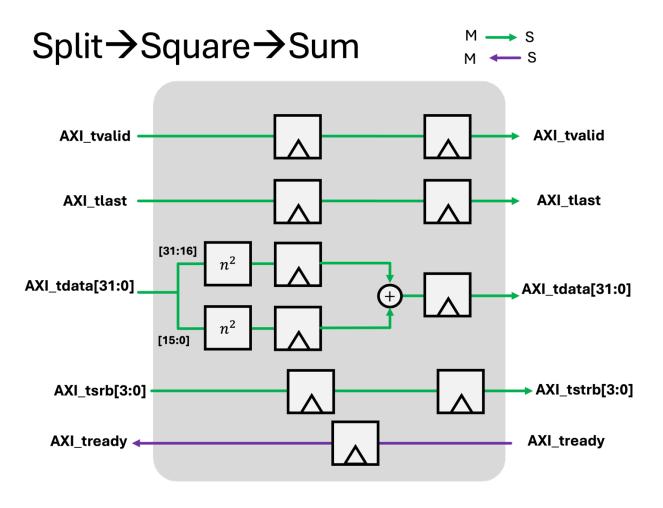
11

Add into a feedback path or something...



Actually we should do

- Add a register on the TREADY pipe
- Any problems with this?



Delaying TREADY

- Delaying the ability to convey a halt (via TREADY) to any upstream device means that there's a delay in stopping that data.
- It has to go somewhere/get absorbed somewhere
- Need a buffer/some sort of very short-form fifo
- You'll hear these called "skid buffers" or "Carloni Buffers"

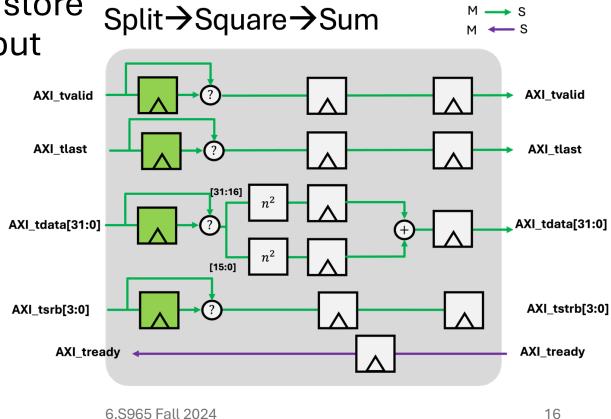
https://ptolemy.berkeley.edu/projects/embedded/research/hsc/class.F02/ee249/lectures/lipClass.pdf

What is a Skid Buffer?

- A device that "eats"/temporarily holds data in the event of the data pipeline having to suddenly slam on the brakes.
- Therefore the system "skids" to a halt.

More complicated than that

 Need something that will selectively let data through or store it based on output



Nice Writeup

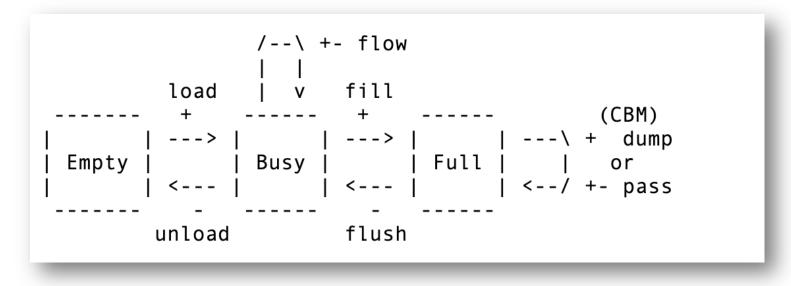
• Kind of an old-school FPGA writeup of a skid buffer found here:

https://fpgacpu.ca/fpga/Pipeline_Skid_Buffer.html

• I wrote my own version based on this discussion. I put up with lecture page for reference.

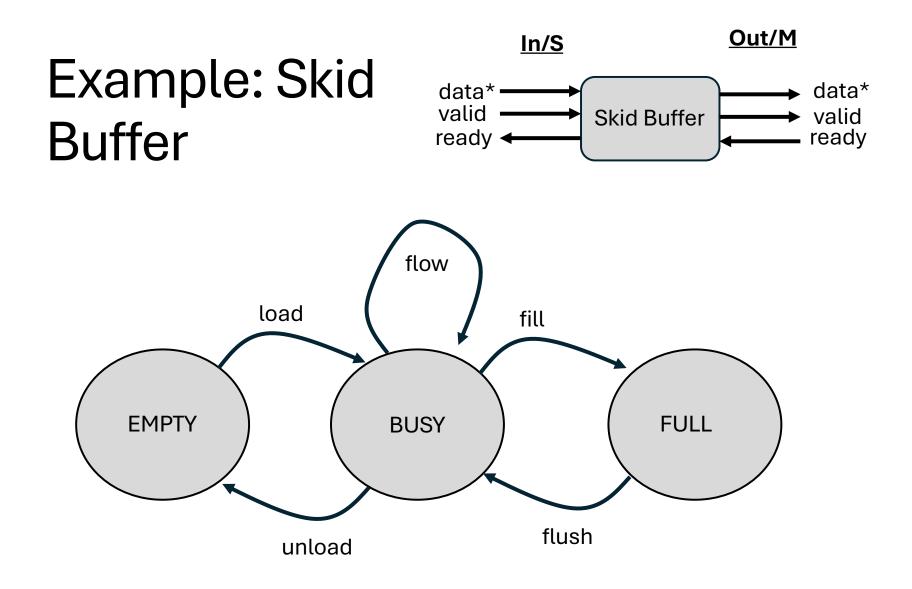
Simple FSM logic

• Three-state FSM can take care of this



https://fpgacpu.ca/fpga/Pipeline_Skid_Buffer.html

6.S965 Fall 2024



Control Path

We separate the control path so the associated data path does not have to know anything about the current state or its encoding.

This FSM assumes the usual meaning and behaviour of valid/ready handshake signals: when both are high, data transfers at the end of the clock cycle. It is an error to raise ready when not able to accept data (thus losing the incoming data), or to raise valid when not able to send data (thus duplicating previously sent data). *These error situations are not handled*.

To operate our datapath as a skid buffer, we need to understand which states we want to allow it to be in, and which state transitions we also allow. This skid buffer has three states:

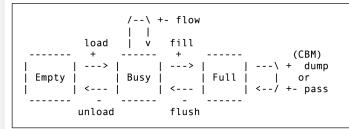
1. It is Empty.

- 2. It is Busy, holding one item of data in the main register, either waiting or actively transferring data through that register.
- 3. It is Full, holding data in both registers, and stopped until the main register is emptied and simultaneously refilled from the buffer register, so no data is lost or reordered. (Without an available empty register, the input interface cannot skid to a stop, so it must signal it is not ready.)
- 4. It is Full and in Circular Buffer Mode, holding data in both registers, and can accept new data into the buffer register while simultaneously replacing the contents of the main register with the current contents of the buffer register.

The operations which transition between these states are:

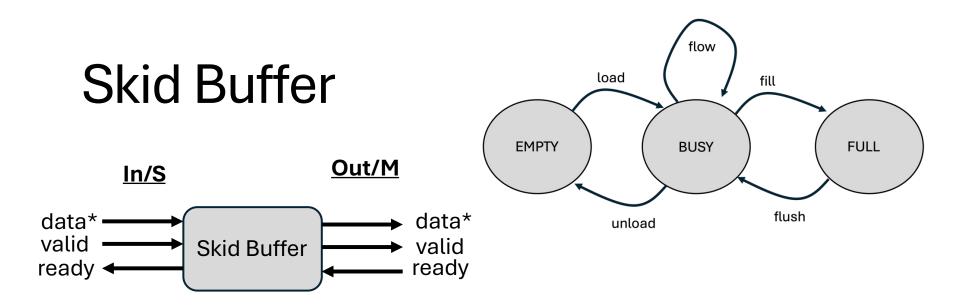
- the input interface inserting a data item into the datapath (+)
- the output interface removing a data item from the datapath (-)
- both interfaces inserting and removing at the same time (+-)

We also descriptively name each transition between states. These names will show up later in the code.



https://fpgacpu.ca/fpga/Pipeline_Skid_Buffer.html

10/21/24



- BUSY is normal operation where data is coming in and out.
- If there's a hiccup on the output side, go to FULL and stall pipeline (s00_tready -> 0)
- If there's a hiccup on the input side, go to EMPTY and stall pipeline (m00_tvalid -> 0)

Skid Buffer

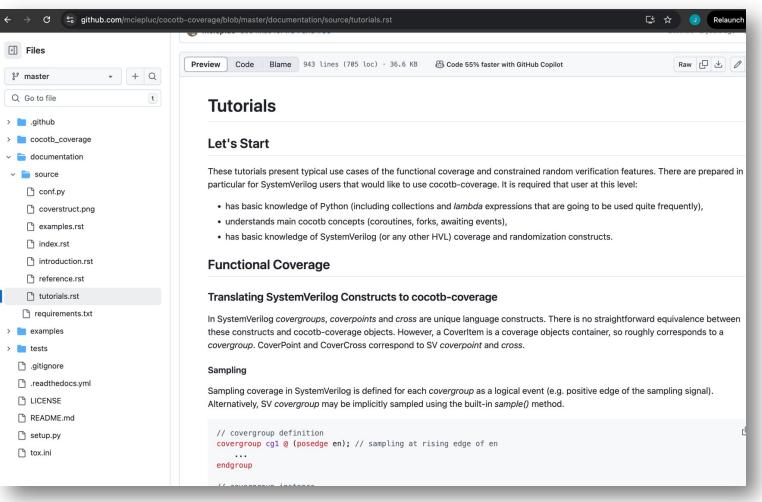
This simple FSM description...glossed over the potential complexity of the implementation: 3 states, each connected to 2 signals (valid/ready) per interface, for a total of 16 possible transitions out of each state, or 48 possible state transitions total.

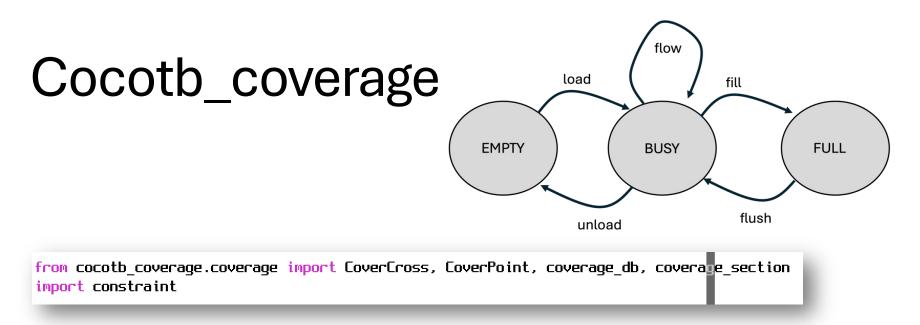
Cocotb Coverage

25 cocotb-coverage.readthedocs.io/en/latest/introduction.html#functional-coverage-with-cocotb-coverage cocotb coverage 1.0 documentation » Introduction Introduction Table of Contents Introduction Functional Coverage in Functional Coverage in SystemVerilog SystemVerilog Functional Coverage with cocotb-coverage In SystemVerilog a fundamental coverage unit is a coverpoint. It contains several bins and each bin may contain Constrained Random several values. Every coverpoint is associated with a variable or signal. At sampling event, the coverpoint vari-Verification Features in SystemVerilog able value is compared with each defined bin. If there is a match, then the number of hits of the particular bin is Constrained Random incremented. Coverpoints are organized in covergroups, which are specific class-like structures. A single cover-Verification Features in group may have several instances and each instance may collect coverage independently. A covergroup requires cocotb-coverage sampling, which may be defined as a logic event (e.g. a positive clock edge). Sampling may also be called im-Previous topic plicitly in the testbench procedural code by invoking a sample() method of the covergroup instance. A bin may be cocotb-coverage also defined as an *ignore_bins*, which means its match does not increase a coverage count, or an *illegal_bins*, which results in error when hit during the test execution. Next topic Reference Documentation Another coverage construct in SystemVerilog is a cross. It automatically generates a Cartesian product of bins from several coverage generation. As it may be diffi-This Page cult or unnecessary to cover all the cross-bins, some of them may be excluded from the analysis. This is possible Show Source using the binsof ... intersect syntax. **Quick search** The most important limitations of the SystemVerilog functional coverage features are: Go • straightforward bins matching criteria - only satisfied by equality or inclusion relation; • bins may be only constants or transitions (possibly wildcard); flat coverage structure – cover groups cannot contain other cover groups, which would correspond better to a verification plan scheme: not possible to get the detailed coverage information in real time (e.g. when a specific bin was hit). Functional Coverage with cocotb-coverage The general assumptions for the architecture of the functional coverage features are as follows: • functional coverage structure should better match a real verification plan; • its syntax should be more flexible, but a separation between coverage and executable code should be maintained: features for analysing the coverage during test execution should be added or extended; • coverage primitives should be able to monitor testbench objects at a higher level of abstraction. The implemented mechanism is based on the idea of decorator design pattern. In Python, a decorator syntax is 6.S965 Fall 2024

10/21/24

Another library with ok docs and source code





- Let's first focus on how we could measure the states that our system exists in?
- This thing has a very clearly defined state machine design and only certain states will connect to certain states

First step is to define some coverage that we care about

 Let's look at current state of our fsm and next/upcoming state of our FSM

CoverPoint

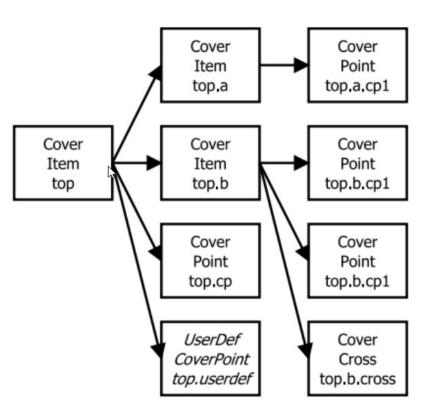
- Object thar represents coverage. Concerned with a signal or combination of signals or state of being.
- Has a name (which you organize in a hierarchical fashion)
- Is used with a function you define
- Qualifies the inputs as one of the values specified in its bins argument

CoverCross

- CoverCross generates the Cartesian Product of multiple CoverPoints
- The CoverCross shown here will have how many possible bins?

Coverage_section

- Is another object that represents a collection of coverpoints (and any related crosses)
- The idea is to hierarchically organize the things you care about



Must Sample/interface with the

actual DUT

- Write a sampling function (just a passthrough here)
- That is then called repeatedly in a monitor that is studying the state/next state on the rising clock edge

Decorator links to coverage_section by name...this is the function that is used by the cover points for analysis

@SC def sampling_function(s,ns): pass

```
async def state_monitor(dut):
   states = {0:'EMPT9', 1:'BU59', 2:'FULL'}
   read_only = ReadOnly() #This is
   falling_edge = FallingEdge(dut.s00_axis_aclk)
   rising_edge = RisingEdge(dut.s00_axis_aclk)
   await read_only
   old_state = dut.state.value
   while True:
      await rising_edge #when module would change
      await read_only
      state = dut.state.value
   sampling_function(states[old_state], states[state])
      old_state = state
```

Then run...

• Launch state monitor here:

tester = SBTester(dut)
tester.start()
cocotb.start_soon(Clock(dut.s00_axis_aclk, 10, units="ns").start())
cocotb.start_soon(state_monitor(dut))

• At end of test...report it out using coverage_db.report_coverage

coverage_db.report_coverage(cocotb.log.info, bins=True)
coverage_file = os.path.join(os.getenv('sim_result', "./"), 'coverage.xml')
coverage_db.export_to_xml(filename=coverage_file)

```
incount = tester.input_mon.stats.received_transactions
outcount = tester.output_mon.stats.received_transactions
assert incount == outcount, f"Transaction Count doesn't match! :/ IN:{incount} vs. OUT: {ou
incount = tester.input_mon.stats.tlast_transactions
outcount = tester.output_mon.stats.tlast_transactions
assert incount == outcount, f"TLAST Transaction Count doesn't match! :/ IN:{incount} vs. OU
raise tester.scoreboard.result
```

The result

top : <cocotb_coverage.coverage.CoverItem object at 0x1029911e0>, coverage=13, size=15 top.st : <cocotb coverage.coverage.CoverItem object at 0x10213d3c0>, coverage=13, size=15 top.st.next state : <cocotb coverage.coverage.CoverPoint object at 0x102991390>, coverage=3, size=3 BIN EMPTY : 212 BIN BUSY : 152 BIN FULL : 307 top.st.state : <cocotb coverage.coverage.CoverPoint object at 0x10213d390>, coverage=12, size=12 **BIN EMPTY : 212** BIN BUSY : 152 BIN FULL : 307 top.st.state.cross : <cocotb coverage.coverage.CoverCross object at 0x10213d360>, coverage=7, size=9 BIN ('EMPTY', 'EMPTY') : 165 BIN ('EMPTY', 'BUSY') : 47 BIN ('EMPT9', 'FULL') : 0 BIN ('BUSY', 'EMPTY') : 47 BIN ('BUSY', 'BUSY') : 103 BIN ('BUS9', 'FULL') : 2 BIN ('FULL', 'EMPTY') : 0 BIN ('FULL', 'BUSY') : 2 BIN ('FULL', 'FULL') : 305 test a passed ** TEST STATUS SIM TIME (ns) REAL TIME (s) RATIO (ns/s) ** PASS 6720.00 0.06 104844.35 ** ** test skid buffer.test a ** TESTS=1 PASS=1 FAIL=0 SKIP=0 6720.00 0.11 63130.17 **

Or if you prefer pretty to read xml

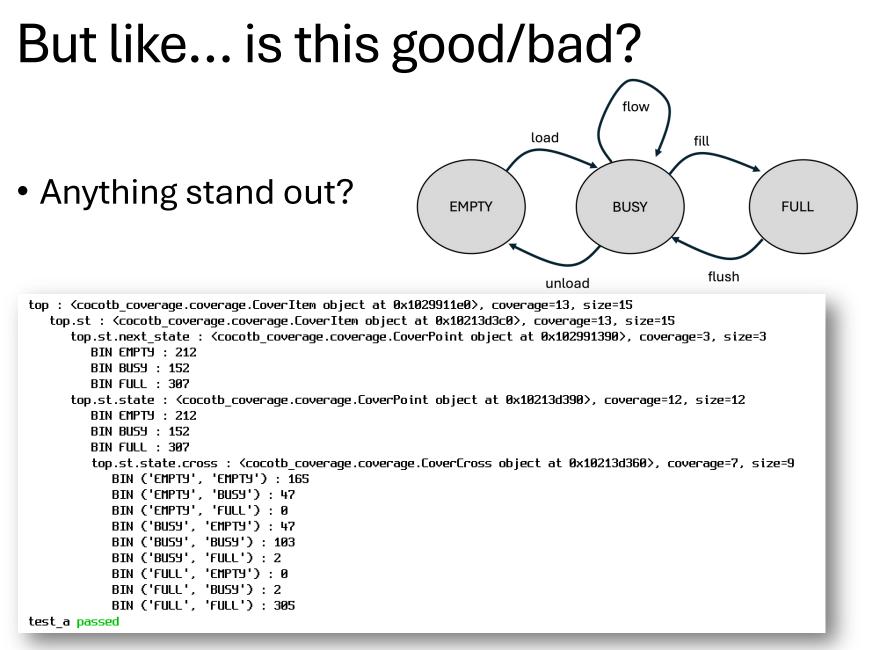
 $\leftarrow \rightarrow C$

file:///Users/jodalyst/cocotb_development/coverage_dev_2/sim/sim_build/coverage.xml

I guess

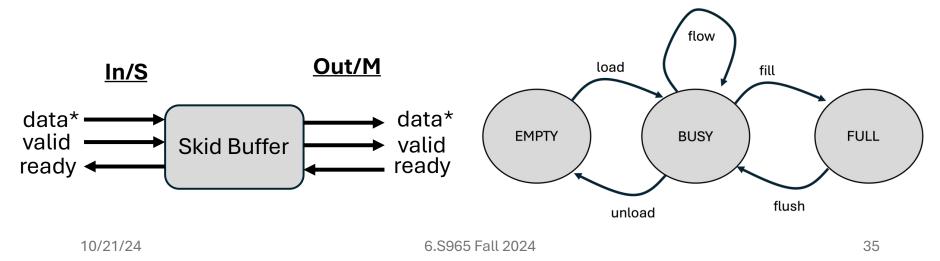
This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
-<top abs_name="top" size="15" coverage="13" cover_percentage="86.67">
  -<st size="15" coverage="13" cover_percentage="86.67" abs_name="top.st">
    -<state size="12" coverage="12" cover_percentage="100.0" abs_name="top.st.state" weight="1" at_least="1">
        <br/>
<bin0 bin="EMPTY" hits="212" abs_name="top.st.state.bin0"/>
        <br/>
<bin1 bin="BUSY" hits="152" abs_name="top.st.state.bin1"/>
        <br/>
<bin2 bin="FULL" hits="307" abs_name="top.st.state.bin2"/>
      -<cross size="9" coverage="7" cover_percentage="77.78" abs_name="top.st.state.cross" weight="1" at least="1">
          <bin0 bin="('EMPTY', 'EMPTY')" hits="165" abs_name="top.st.state.cross.bin0"/>
          <bin1 bin="('EMPTY', 'BUSY')" hits="47" abs_name="top.st.state.cross.bin1"/>
          <bin2 bin="('EMPTY', 'FULL')" hits="0" abs_name="top.st.state.cross.bin2"/>
          <bin3 bin="('BUSY', 'EMPTY')" hits="47" abs_name="top.st.state.cross.bin3"/>
          <bi><bin4 bin="('BUSY', 'BUSY')" hits="103" abs name="top.st.state.cross.bin4"/>
          <bin5 bin="('BUSY', 'FULL')" hits="2" abs_name="top.st.state.cross.bin5"/>
          <bin6 bin="('FULL', 'EMPTY')" hits="0" abs_name="top.st.state.cross.bin6"/>
          <br/>
<bin7 bin="('FULL', 'BUSY')" hits="2" abs_name="top.st.state.cross.bin7"/>
          <bin8 bin="('FULL', 'FULL')" hits="305" abs_name="top.st.state.cross.bin8"/>
        </cross>
     </state>
    -<next state size="3" cover age="3" cover percentage="100.0" abs_name="top.st.next state" weight="1" at least="1">
        <br/>
<bin0 bin="EMPTY" hits="212" abs_name="top.st.next_state.bin0"/>
        <br/>
<bin1 bin="BUSY" hits="152" abs_name="top.st.next_state.bin1"/>
        <br/>
<bin2 bin="FULL" hits="307" abs_name="top.st.next_state.bin2"/>
     </next state>
   </st>
 </top>
```



This is kinda scary actually.

This simple FSM description...glossed over the potential complexity of the implementation: 3 states, each connected to 2 signals (valid/ready) per interface, for a total of 16 possible transitions out of each state, or 48 possible state transitions total.



So let's do state and input

- Come up with STS covergroup (State and Signals)
- I want to look at the different states of my module as well as its exposure to different signal combinations on both S00 and M00 side

```
STS = coverage section(
CoverPoint("top.st sig.state",
            xf=lambda state,sig: state,
            bins=['EMPT9', 'BUS9', 'FULL']
            ).
CoverPoint("top.st sig.s00 tvalid",
            xf=lambda state,sig: sig.get('s00 tvalid'),
            bins=[True, False]
            ).
CoverPoint("top.st sig.s00 tready",
            xf=lambda state,sig: sig.get('s00 tready'),
            bins=[True, False]
            ).
CoverPoint("top.st sig.m00 tvalid",
            xf=lambda state,sig: sig.get('m00 tvalid'),
            bins=[True, False]
            ),
CoverPoint("top.st sig.m00 tready",
            xf=lambda state,sig: sig.get('m00 tready'),
            bins=[True, False]
            ).
CoverCross("top.st sig.cross",
            items=[ "top.st sig.state",
                    "top.st sig.s00 tvalid".
                    "top.st sig.s00 tready".
                    "top.st sig.m00 tvalid".
                    "top.st sig.m00 tready"]
            )
)
```

Write a little monitor coroutine

- Runs and checks the state and input signals going into every rising edge...
- Puts it in a nice dictionary and I hand things off to the coverage function

@STS def sts_sampling_function(state,sig): pass

```
async def sts_monitor(dut):
   states = {0:'EMPTY', 1:'BUSY', 2:'FULL'}
   read_only = ReadOnly()
   falling_edge = FallingEdge(dut.s00_axis_aclk)
   rising_edge = RisingEdge(dut.s00_axis_aclk)
   await read_only
   old state = dut.state.value
```

while True:

```
await falling_edge #when module would change
await read_only
state = dut.state.value
sig = {'s00_tvalid':dut.s00_axis_tvalid.value,
        's00_tready':dut.s00_axis_tready.value,
        'm00_tvalid':dut.m00_axis_tvalid.value,
        'm00_tready':dut.m00_axis_tready.value
}
sts_sampling_function(states[state],sig)
```

So we run....

/test_skid_buffer.py:143: DeprecationWarning: Use `bv.integer` instead.	From before
top : <cocotb_coverage.coverage.coveritem 0x10711d270="" at="" object="">, coverage=30</cocotb_coverage.coverage.coveritem>	, size=74
top.st : <cocotb_coverage.coverage.coveritem 0x1068c80a0="" at="" object="">, cover</cocotb_coverage.coverage.coveritem>	age=13, size=15
top.st.next_state : <cocotb_coverage.coverage.coverpoint 0x10<="" at="" object="" td=""><td>711d420>, coverage=3, size=3</td></cocotb_coverage.coverage.coverpoint>	711d420>, coverage=3, size=3
BIN EMPTY : 212	
BIN BUS9 : 152	
BIN FULL : 307	
top.st.state : <cocotb_coverage.coverage.coverpoint 0x1068c80<="" at="" object="" td=""><td>70>, coverage=12, size=12</td></cocotb_coverage.coverage.coverpoint>	70>, coverage=12, size=12
BIN EMPTY : 212	
BIN BU59 : 152	
BIN FULL : 307	
top.st.state.cross : <cocotb_coverage.coverage.covercross at<="" object="" td=""><td>0x1068c8040>, coverage=7, size</td></cocotb_coverage.coverage.covercross>	0x1068c8040>, coverage=7, size
BIN ('EMPTY', 'EMPTY') : 165	
BIN ('EMPTY', 'BUSY') : 47	
BIN ('EMPTY', 'FULL') : 0	
BIN ('BUSY', 'EMPTY') : 47	
BIN ('BUSY', 'BUSY') : 103	
BIN ('BUSY', 'FULL') : 2	
BIN ('FULL', 'EMPTY') : 0	N Laure
BIN ('FULL', 'BUSY') : 2	New
BIN ('FULL', 'FULL') : 305	
top.st_sig : <cocotb_coverage.coverage.coveritem 0x10711dab0="" at="" object="">, c</cocotb_coverage.coverage.coveritem>	2
top.st_sig.cross : <cocotb_coverage.coverage.covercross 0x107<="" at="" object="" td=""><td>11e170>, coverage=6, size=48</td></cocotb_coverage.coverage.covercross>	11e170>, coverage=6, size=48
BIN ('CHPT9', True, True, True, True) : 0	
BIN ('EMPTY', True, True, True, False) : 0	
BIN ('EMPTY', True, True, False, True) : 0	
BIN ('EMPTY', True, True, False, False) : 0	
BIN ('EMPTY', True, False, True, True) : 0	
BIN ('EMPTY', True, False, True, False) : 0	
BIN ('EMPTY', True, False, False, True) : 0	
BIN ('EMPTY', True, False, False, False) : 0	
BIN ('EMPTY', False, True, True, True) : 0	
BIN ('EMPTY', False, True, True, False) : 0	
BIN ('EMPTY', False, True, False, True) : 212	
BIN ('EMPTY', False, True, False, False) : 0	
BIN ('EMPTY', False, False, True, True) : 0	
BIN ('EMPTY', False, False, True, False) : 0	
BIN ('EMPTY', False, False, False, True) : 0	
BIN ('EMPTY', False, False, False, False) : 0	
BIN ('BUSY', True, True, True, True) : 149	
BIN ('BUSY', True, True, True, False) : 1	
BIN ('BUSY', True, True, False, True) : 0	

Very Limited Coverage (12.5%)

$\leftarrow \ \ \rightarrow \ \ C$ file:///Users/jodalyst/cocotb_development/coverage_dev_2/sim/sim_build/coverage.xml
<pre> <</pre>
 sin1 bin="BUSY" hits="152" abs name="top.st.next state.bin1"/>
 stin2 bin="FULL" hits="307" abs_name="top.st.next_state.bin2"/>
- <st_sig abs_name="top.st_sig" cover_percentage="28.81" coverage="17" size="59"></st_sig>
- <state abs_name="top.st_sig.state" at_least="1" cover_percentage="100.0" coverage="3" size="3" weight="1"></state>
 bin0 bin="EMPTY" hits="212" abs_name="top.st_sig.state.bin0"/>
 bin1 bin="BUSY" hits="152" abs_name="top.st_sig.state.bin1"/>
 bin2 bin="FULL" hits="307" abs_name="top.st_sig.state.bin2"/>
- <s00_tvalid abs_name="top.st_sig.s00_tvalid" at_least="1" cover_percentage="100.0" coverage="2" size="2" weight="1"></s00_tvalid>
 bin9 bin="True" hits="455" abs_name="top.st_sig.s00_tvalid.bin0"/>
<bin1 abs_name="top.st_sig.s00_tvalid.bin1" bin="False" hits="216"></bin1>
- <s00_tready abs_name="top.st_sig.s00_tready" at_least="1" cover_percentage="100.0" coverage="2" size="2" weight="1"></s00_tready>
 bin="True" hits="364" abs_name="top.st_sig.s00_tready.bin0"/>
<bin1 abs_name="top.st_sig.s00_tready.bin1" bin="False" hits="307"></bin1>
- <m00_tvalid abs_name="top.st_sig.m00_tvalid" at_least="1" cover_percentage="100.0" coverage="2" size="2" weight="1"></m00_tvalid>
<bin0 abs_name="top.st_sig.m00_tvalid.bin0" bin="True" hits="459"></bin0>
 sig.m00_tvalid.bin1"/>
- <m00_tready abs_name="top.st_sig.m00_tready" at_least="1" cover_percentage="100.0" coverage="2" size="2" weight="1"></m00_tready>
 sin0 bin="True" hits="361" abs_name="top.st_sig.m00_tready.bin0"/> hit=1 bin="True" hits="361" abs_name="top.st_sig.m00_tready.bin0"/>
- <cross abs="" at="" cover="" coverage="6" least="1" name="top.st sig.cross" percentage="12.5" size="48" weight="1"></cross>
<pre>-<cross abs_name="top.st_sig.cross" at_reast="1" cover_percentage="12.5" coverage="6" size="46" weight="1"></cross></pre>
<pre>chin1 bin="('EMPTY', True, True, True, False)" bits="0" abs_name="top.st_sig.cross.bin1"/></pre>
<pre><bin2 ('empty',="" abs_name="top.st_sig.cross.bin2" bin2="" false,="" hits="0" true)"="" true,=""></bin2></pre>
<pre><bin3 abs_name="top.st_sig.cross.bin3" bin="('EMPTY', True, True, False, False)" hits="0"></bin3></pre>
<pre><bid></bid></pre> <pre><bid></bid></pre> <pre><bid></bid></pre> <pre><bid></bid></pre> <pre><bid></bid></pre> <pre><bid></bid></pre> <pre><bid></bid></pre> <pre></pre> <pre><bid></bid></pre> <pre> <bid></bid></pre> <pre> <bid></bid></pre> <pre> <bid></bid></pre> <pre> <bid></bid></pre> <pre> <bid></bid></pre> <pre> <bid></bid></pre> <pre> <bid></bid></pre> <pre> <bid></bid></pre> <pre> <bid></bid></pre> <pre> <bid></bid></pre> <pre> <bid></bid></pre> <pre> <br <="" td=""/></pre>
<pre><bins abs_name="top.st_sig.cross.bin5" bins="('EMPTY', True, False, True, False)" hits="0"></bins></pre>
Abia Che MENONY The Flor Flor The Meter 100 abs and the transfer to a second bia che in the second bia che flor the The Meter 100 abs and the second bia constraints of the

Looking Closer...

top.st sig.cross : <cocotb coverage.coverage.CoverCross object at 0x10711e170>, coverage=6, size=48 BIN ('EMPTY', True, True, True, True) : 0 BIN ('EMPTY', True, True, True, False) : 0 BIN ('EMPTY', True, True, False, True) : 0 BIN ('EMPTY', True, True, False, False) : 0 BIN ('EMPTY', True, False, True, True) : 0 BIN ('EMPTY', True, False, True, False) : 0 BIN ('EMPTY', True, False, False, True) : 0 BIN ('EMPTY', True, False, False, False) : 0 BIN ('EMPTY', False, True, True, True) : 0 BIN ('EMPTY', False, True, True, False) : 0 BIN ('EMPTY', False, True, False, True) : 212 BIN ('EMPTY', False, True, False, False) : 0 BIN ('EMPTY', False, False, True, True) : 0 BIN ('EMPTY', False, False, True, False) : 0 BIN ('EMPTY', False, False, False, True) : 0 BIN ('EMPTY', False, False, False, False) : 0 BIN ('BUSY', True, True, True, True) : 149 BIN ('BUSY', True, True, True, False) : 1 BIN ('BUSY', True, True, False, True) : 0 BIN ('BUSY', True, True, False, False) : 0 BIN ('BUSY', True, False, True, True) : 0 BIN ('BUSY', True, False, True, False) : 0 BIN ('BUSY', True, False, False, True) : 0 BIN ('BUSY', True, False, False, False) : 0 BIN ('BUSY', False, True, True, True) : 0 BIN ('BUSY', False, True, True, False) : 2 BIN ('BUSY', False, True, False, True) : 0 BIN ('BUSY', False, True, False, False) : 0 BIN ('BUSY', False, False, True, True) : 0 BIN ('BUSY', False, False, True, False) : 0 BIN ('BUSY', False, False, False, True) : 0 BIN ('BUSY', False, False, False, False) : 0 BIN ('FULL', True, True, True, True) : 0 BIN ('FULL', True, True, True, False) : 0 BIN ('FULL', True, True, False, True) : 0 BIN ('FULL', True, True, False, False) : 0 BIN ('FULL', True, False, True, True) : 0 BIN ('FULL', True, False, True, False) : 305 BIN ('FULL', True, False, False, True) : 0 BIN ('FULL', True, False, False, False) : 0 BIN ('FULL', False, True, True, True) : 0 BIN ('FULL', False, True, True, False) : 0 BIN ('FULL', False, True, False, True) : 0 BIN ('FULL', False, True, False, False) : 0 BIN ('FULL', False, False, True, True) : 0 BIN ('FULL', False, False, True, False) : 2 BIN ('FULL', False, False, False, True) : 0 BIN ('FULL', False, False, False, False) : 0

 Very little of the state of possibility was covered here.

10/21/24

So what do we do?

- Instead of having long bursts of ready and then long bursts of !ready
- Maybe randomize it?

```
#feed the driver:
for i in range(50):
 data = {'type':'single', "contents":{"data": random.randint(1,255),"last":
  tester.input driver.append(data)
#data = {'type':'burst', "contents":{"data": np.array(20*[0]+[1]+30*[0]+[-2]
data = {'type':'burst', "contents":{"data": np.array(list(range(100)))}}
tester.input driver.append(data)
for x in range(1000):
 await ClockCycles(dut.s00 axis aclk,1)
 coin flip = random.random()>0.1
  if coin flip:
    await set_ready(dut,1)
  else:
    await set ready(dut,0)
#await ClockCycles(dut.s00 axis aclk, 50)
#await set ready(dut,0)
#await ClockEycles(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 ClockEycles(dut.s00 axis aclk, 300)
```

Now?

 Better than before for sure

• 22%

Of course also keep making sure it passes the scoreboard checks

> cocotb.regression cocotb.regression

10/21/24

	BIN (ENFIS , Irue, Irue, False, False) : 4
	BIN ('EMPTY', True, False, True, True) : 0
)	BIN ('EMPTY', True, False, True, False) : 0
	BIN ('EMPTY', True, False, False, True) : 0
	BIN ('EMPTY', True, False, False, False) : 0
	BIN ('EMPTY', False, True, True, True) : 0
	BIN ('EMPTY', False, True, True, False) : 0 BIN ('EMPTY', False, True, False, True) : 989
	BIN ('EMPTY', False, True, False, False) : 89
	BIN ('EMPTY', False, False, True, True) : 0
	BIN ('EMPTY', False, False, True, False) : 0
	BIN ('EMPTY', False, False, False, True) : 0
n	BIN ('EMPTY', False, False, False, False) : 0
	BIN ('BUSY', True, True, True, True) : 83
CUIRO	BIN ('BUSY', True, True, True, False) : 22
sure	BIN ('BUSY', True, True, False, True) : 0
	BIN ('BUSY', True, True, False, False) : 0
	BIN ('BUSY', True, False, True, True) : 0
	BIN ('BUSY', True, False, True, False) : 0
	BIN ('BUSY', True, False, False, True) : 0 BIN ('BUSY', True, False, False, False) : 0
	BIN ('BUSY', False, True, True, True) : 45
	BIN ('BUSY', False, True, False) : 4
	BIN ('BUSY', False, True, False, True) : 0
	BIN ('BUSY', False, True, False, False) : 0
	BIN ('BUSY', False, False, True, True) : 0
	BIN ('BUSY', false, false, True, false) : 0
ep makin	of BIN ('BUSY', False, False, True) : 0
opmanna	
e scoreboa	BIN ('FULL', True, True, True): 0
30016000	
	BIN ('FULL', True, True, False, True) : 0
	BIN ('FULL', True, True, False, False) : 0
	BIN ('FULL', True, False, True, True) : 20 BIN ('FULL', True, False, True, False) : 3
	BIN ('FULL', True, False, False, True) : 0
	BIN ('FULL', True, False, False, False) : 0
	BIN ('FULL', False, True, True, True) : 0
	BIN ('FULL', False, True, True, False) : 0
	DTM ('EUU' Falca Tous Falca Tous) • 0
	test_a passed
¥	***************************************
¥	↔ TEST STATUS SIM TIME (ns) REAL TIME (s) RATIO (ns/s) **
¥	· · · · · · · · · · · · · · · · · · ·
¥	ε test skid buffer.test a PR55 13030.00 0.17 74699.18 ★★

	↔ TESTS=1 PASS=1 FAIL=0 SKIP=0 13030.00 0.22 59824.71 **
	<pre></pre>
×	
×	6.S965 Fall 2024 42

top.st_sig : <cocotb_coverage.coverage.CoverItem object at 0x1049e1b10>, coverage=22, size=59

BIN ('EMPTY', True, True, True, True) : 0 BIN ('EMPTY', True, True, True, False) : 0 BIN ('EMPTY', True, True, False, True) : 41 BIN ('EMPTY', True, True, False, False) : 4

top.st_sig.ccoss : https://www.coverage.foverfross-object-at-0x1049e21d0>, coverage=11, size=48

Ignore the Driver

Just throw crap at this system

def rando_assign(signal, size):
 if random.random()>0.5:
 signal.value = random.randint(0,2**size-1)
 else:
 signal.value = 0

```
@cocotb.test()
async def test a(dut):
   """cocotb test for averager controller""
   tester = SBTester(dut)
   tester.start()
   cocotb.start_soon(Clock(dut.s00 axis_aclk, 10, units="ns").start())
   cocotb.start soon(state monitor(dut))
   cocotb.start soon(sts monitor(dut))
   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"
   # tester.input driver.append(data)
   ##data = {'type':'burst', "contents":{"data": np.array(20*[0]+[1]+30*[0]+[-2]
   #data = {'type':'burst', "contents":{"data": np.array(list(range(100)))}}
   #tester.input driver.append(data)
    for x in range(1000):
     await FallingEdge(dut.s00 axis aclk)
     rando assign(dut.s00 axis tvalid,1)
     rando assign(dut.s00 axis tlast,1)
     rando assign(dut.s00 axis tdata,32)
     rando assign(dut.m00 axis tready,1)
   #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)
```

Resulting Waveform

C_M00_AXIS_TDATA_WIDTH	00000020
C_S00_AXIS_TDATA_WIDTH	(0000020
data_buffer_wren	
data_out_wren	
fill	
flow	
flush	
insert	
load	
remove	
m00_axis_aclk	
m00_axis_aresetn	
m00_axis_tdata [31:0]	
m00_axis_tlast	
m00_axis_tvalid	
m00_axis_tready	
m00_axis_tstrb [3:0]	0
s00_axis_aclk	
s00_axis_aresetn	
s00_axis_tdata [31:0]	1290549872
s00_axis_tvalid	
s00_axis_tready	
state [31:0]	
s00_axis_tlast	<u>i 1900 - 19 a 19</u>
s00_axis_tstrb [3:0]	0
tdata_buffer [31:0]	(@
tlast_buffer	
tstrb_buffer [3:0]	0
unload	
use_buffered_data	

God abandoned this testbench

But at the same time...

- You only need to dig into that testbench if you see errors
- And it actually seems to be responding ok

Slightly improved coverage

 And stuff still passes so that's good at least

BIN FULL : 304

** test skid buffer.test a

** TESTS=1 PASS=1 FAIL=0 SKIP=0

test_a passed

****** TEST

top.st sig : <cocotb coverage.coverage.CoverItem object at 0x106b25990>, coverage=23, size=59 top.st_sig.cross : <cocotb_coverage.coverage.CoverCross object at 0x106b26050>, coverage=12, size=48 BIN (EMPIS, Irue, Irue, Irue, Irue) : 0 BIN ('EMPTY', True, True, True, False) : 0 BIN ('EMPTY', True, True, False, True) : 18 BIN ('EMPTY', True, True, False, False) : 65 BIN ('EMPTY', True, False, True, True) : 0 BIN ('EMPTY', True, False, True, False) : 0 BIN ('EMPTY', True, False, False, True) : 0 BIN ('EMPTY', True, False, False, False) : 0 BIN ('EMPTY', False, True, True, True) : 0 BIN ('EMPT9', False, True, True, False) : 0 BIN ('EMPTY', False, True, False, True) : 348 BIN ('EMPTY', False, True, False, False) : 164 BIN ('EMPTY', False, False, True, True) : 0 BIN ('EMPTY', False, False, True, False) : 0 BIN ('EMPTY', False, False, False, True) : 0 BIN ('EMPTY', False, False, False, False) : 0 BIN ('BUSY', True, True, True, True) : 20 BIN ('BUSY', True, True, True, False) : 76 BIN ('BUSY', True, True, False, True) : 0 BIN ('BUSY', True, True, False, False) : 0 BIN ('BUSY', True, False, True, True) : 0 BIN ('BUSY', True, False, True, False) : 0 BIN ('BUSY', True, False, False, True) : 0 BIN ('BUSY', True, False, False, False) : 0 BIN ('BUSY', False, True, True, True) : 83 BIN ('BUSY', False, True, True, False) : 223 BIN ('BUSY', False, True, False, True) : 0 BIN ('BUSY', False, True, False, False) : 0 BIN ('BUSY', False, False, True, True) : 0 BIN ('BUSY', False, False, True, False) : 0 BIN ('BUSY', False, False, False, True) : 0 BIN ('BUSY', False, False, False, False) : 0 BIN ('FULL', True, True, True, True) : 0 BIN ('FULL', True, True, True, False) : 0 BIN ('FULL', True, True, False, True) : 0 BIN ('FULL', True, True, False, False) : 0 BIN ('FULL', True, False, True, True) : 19 BIN ('FULL', True, False, True, False) : 53 BIN ('FULL', True, False, False, True) : 0 STATUS SIM TIME (ns) REAL TIME (s) RATIO (ns/s) ** PASS 13020.00 0.18 72065.18 **

10/21/24

6.S965 Fall 2024

0.22

58280.34 ******

13020.00

	top.st_sig : <cocotb_coverage.coverage.coveritem 0x106ce9b40="" at="" object="">, coverage=23, size=59 top.st_sig.cross : <cocotb_coverage.coverage.covercross 0x106cea200="" at="" object="">, coverage=12, size=48</cocotb_coverage.coverage.covercross></cocotb_coverage.coverage.coveritem>	
	BIN ('ENPI9', Irue, Irue, Irue, Irue) : 0 BIN ('EMPT9', True, True, True, False) : 0	
	BIN ('EMPTY', True, True, False, True) : 178	
Run it more?	BIN ('EMPTY', True, False, False): 541	
	BIN ('EMPTY', True, False, True, True) : 0 BIN ('EMPTY', True, False, True, False) : 0	
	BIN ('EIPTY', True, False, False, True) : 0	
	BIN ('EMPTY', True, False, False, False) : 0	
	BIN ('EMPTY', False, True, True, True) : 0	
	BIN ('CMPTS', False, True, True, False) : 0	
	BIN ('EMPTY', False, True, False, True) : 854 BIN ('EMPTY', False, True, False, False) : 1645	
	BIN ('EITP'), False, False, True, True) : 0	
T , , , ,	BIN ('EMPTY', False, False, True, False) : 0	
 Try to catch some 	BIN ('EMPTY', false, false, false, True) : 0	
ing to outon some	BIN ('EMPTY', False, False, False, False) : 0	
	BIN ('BUSY', True, True, True, True) : 233	
other things?	BIN ('BUSY', True, True, True, False) : 761 BIN ('BUSY', True, True, False, True) : 0	
other things?	BIN (BUSY), True, True, False, False): 0	
Ŭ	BIN ('BU59', True, False, True, True) : 0	
	BIN ('BUSY', True, False, True, False) : 0	
	BIN ('BUSY', True, False, False, True) : 0	
##data = {'type':'burst', "contents":{"data": np.ar	rau(20×[0]+[1]+30×[0 ^{ie} , false, false) : 0	
#data = {'type':'burst', "contents":{"data": np.arr	ag(list(range(100))) _{e, false, True}): 0	
<pre>#tester.input driver.append(data)</pre>	e, False, False) : 0	
for x in range(10000):	se, True, True) : 0	
	se, True, False) : 0	
await FallingEdge(dut.s00 axis aclk)	.se, False, True) : 0 .se, False, False) : 0	
rando assign(dut.s00 axis tvalid,1)	, True, True): 0	
	:, True, False) : 0	
rando_assign(dut.s00_axis_tlast,1)	, False, True) : 0	
rando assign(dut.s00 axis tdata, <mark>32</mark>)	:, False, False) : 0 ;e, True, True) : 190	
	ie, True, False) : 563	
rando_assign(dut.m00_axis_tready, 1)	ie, False, True): 0	
	ie, False, False) : 0	
	BIN ('FULL', False, True, True) : 0	
	BIN ('FULL', False, True, True, False) : 0 BIN ('FULL', False, True, False, True) : 0	
	BIN (FULL', False, True, False, False) : 0	
	BIN ('FULL', False, False, True, True) : 571	
	BIN ('FULL', false, false, True, false) : 1711	
-	BIN ('FULL', False, False, False, True) : 0	
Soome to cap out	BIN ('FULL', False, False, False, False) : 0	
• Seems to cap out		
•		

What Else?

- It'd be nice to be able to see how many input patterns we got of distinct shapes/types in a higher level labeling
- It'd also be good to have more flexibility with the math/randomization/and/or have the randomization focus on edge cases rather than just really random numbers
- Also some things don't have "state" so you may need to characterize off of just external showing signals