6.S965
Digital Systems Laboratory II

Lecture 11

Administrative Stuff

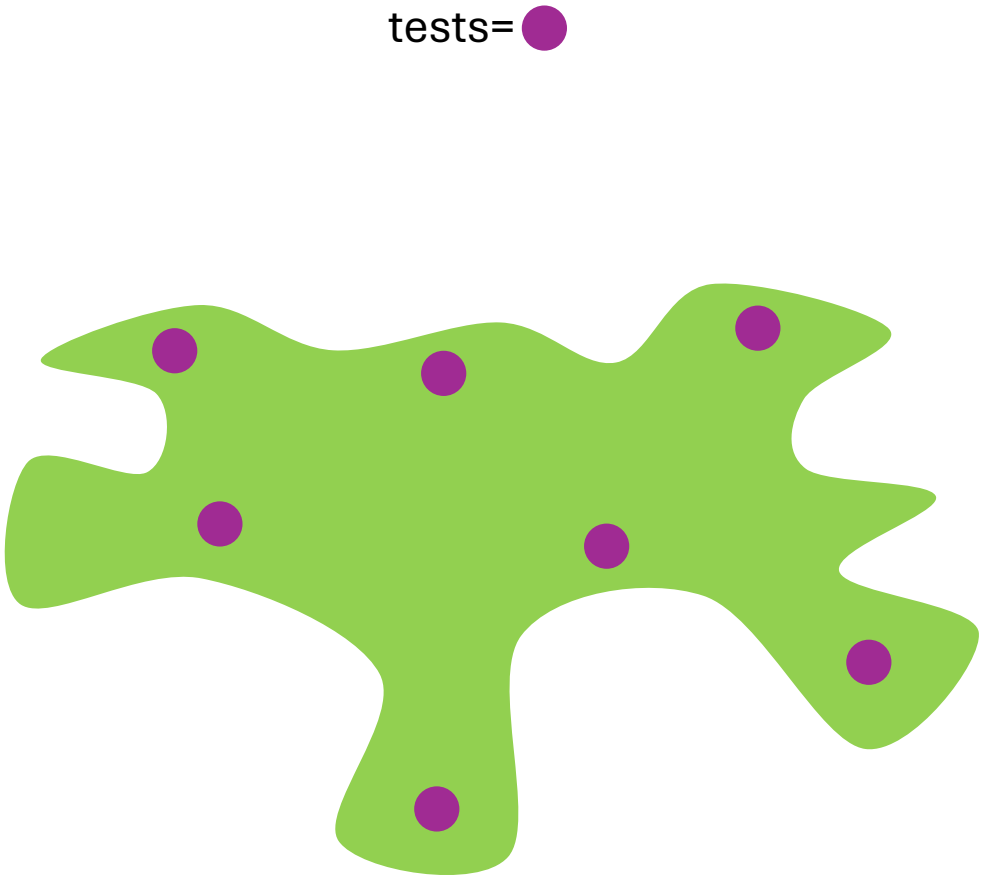
- Last week of stuff (week 6) content was released Friday
- Try to do by the end of the week.
- No idea why the RFSoC samples at half the specified rate.
  - Anybody figure that out they'll get three US Dollars
- I'm going through projects and things now. Sorry 6.205 delayed me. Feel free to also reach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

Coverage

What is it? What does it mean?

Coverage

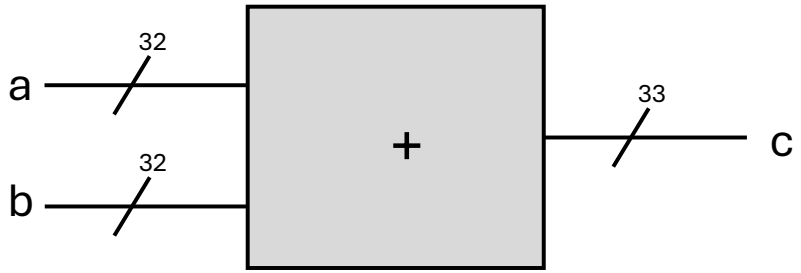
- Is concerned with how much have you tested a DUT



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.

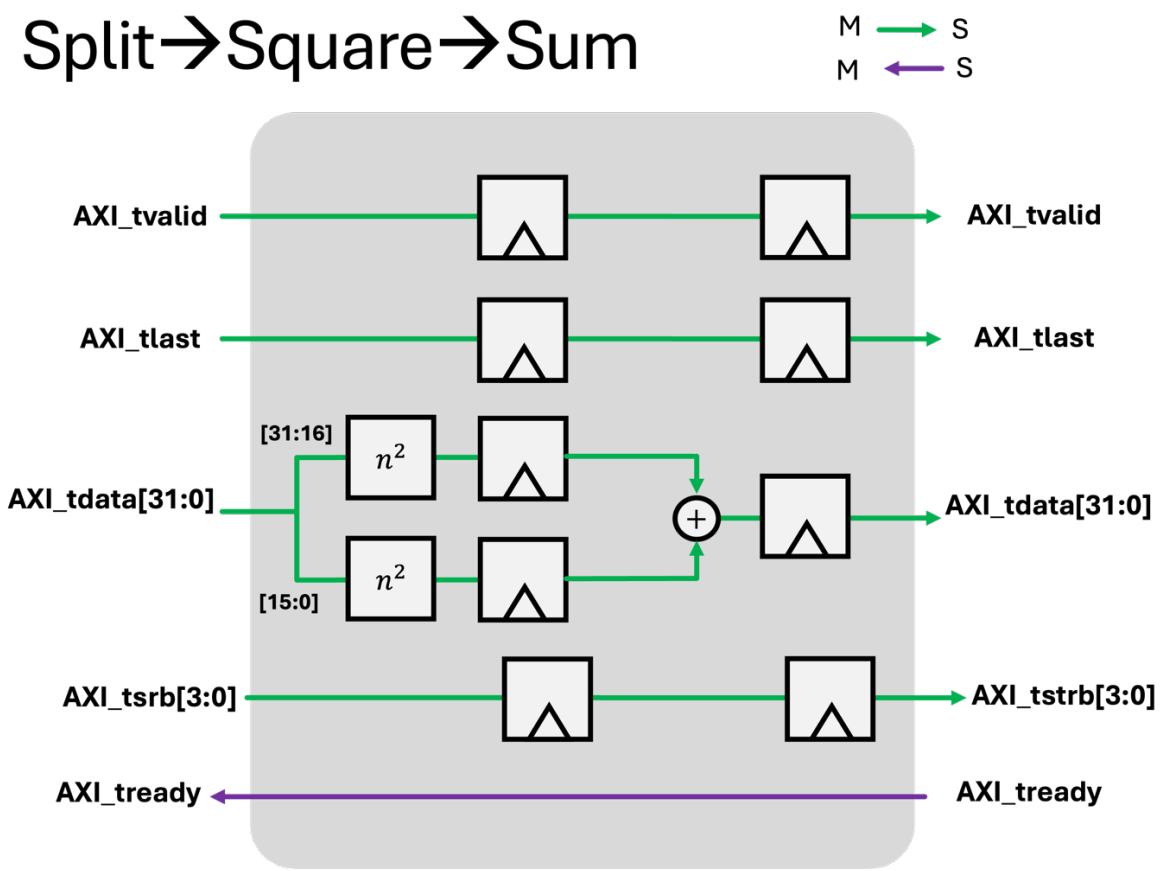
So let’s look at an example…

- We’ll revisit the issue of TREADY propagation and build a module to handle that properly.

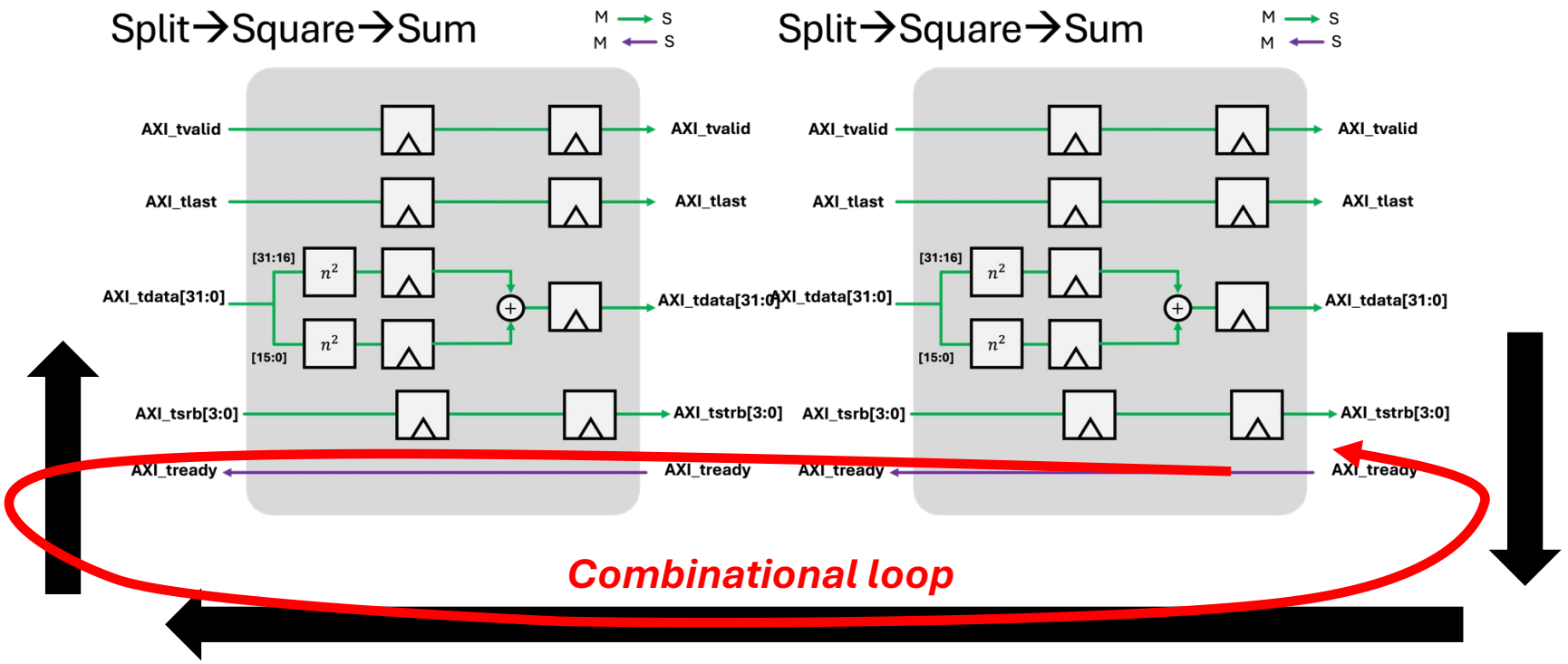
- This plagued some people in earlier weeks.

Week 4 Split-Square-Sum

- Any Problems?

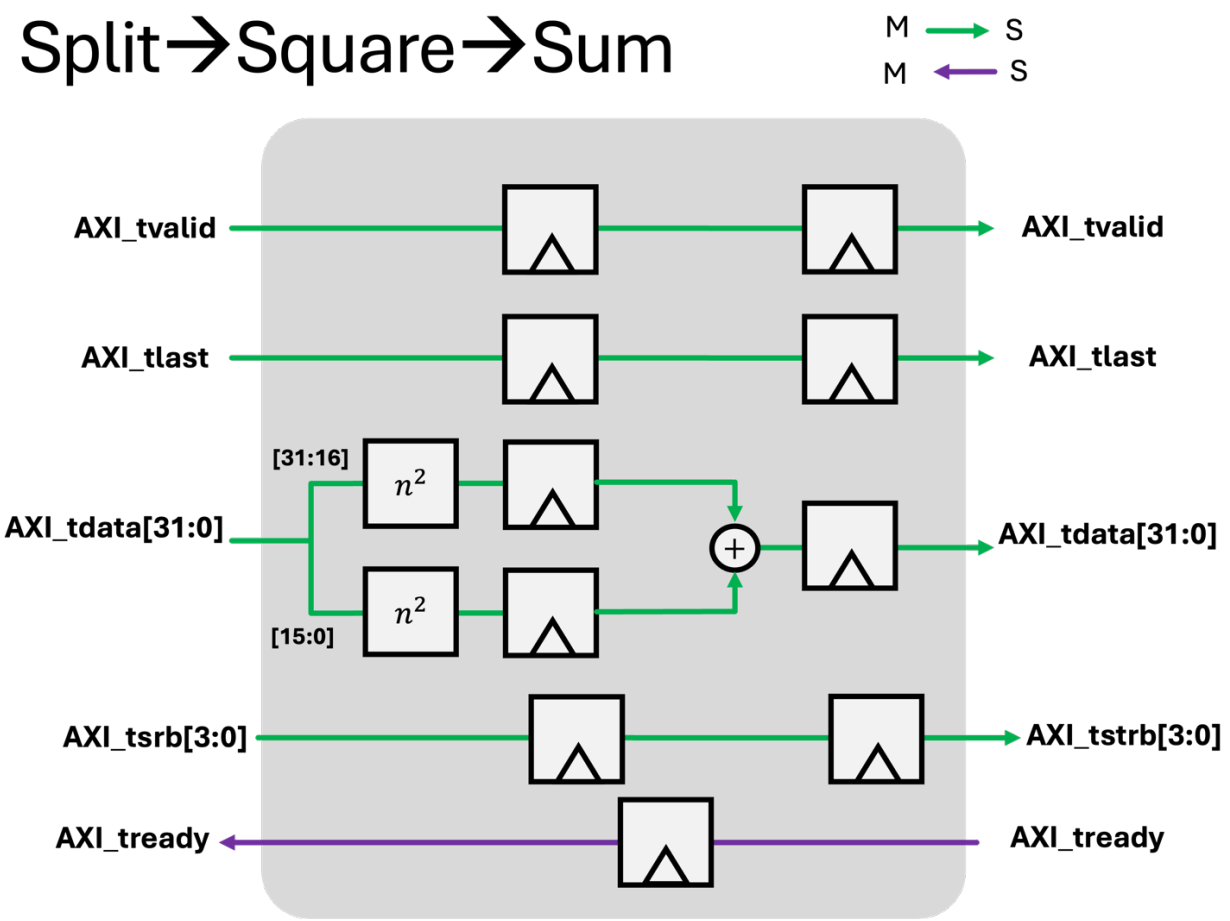


Add into a feedback path or something…



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"

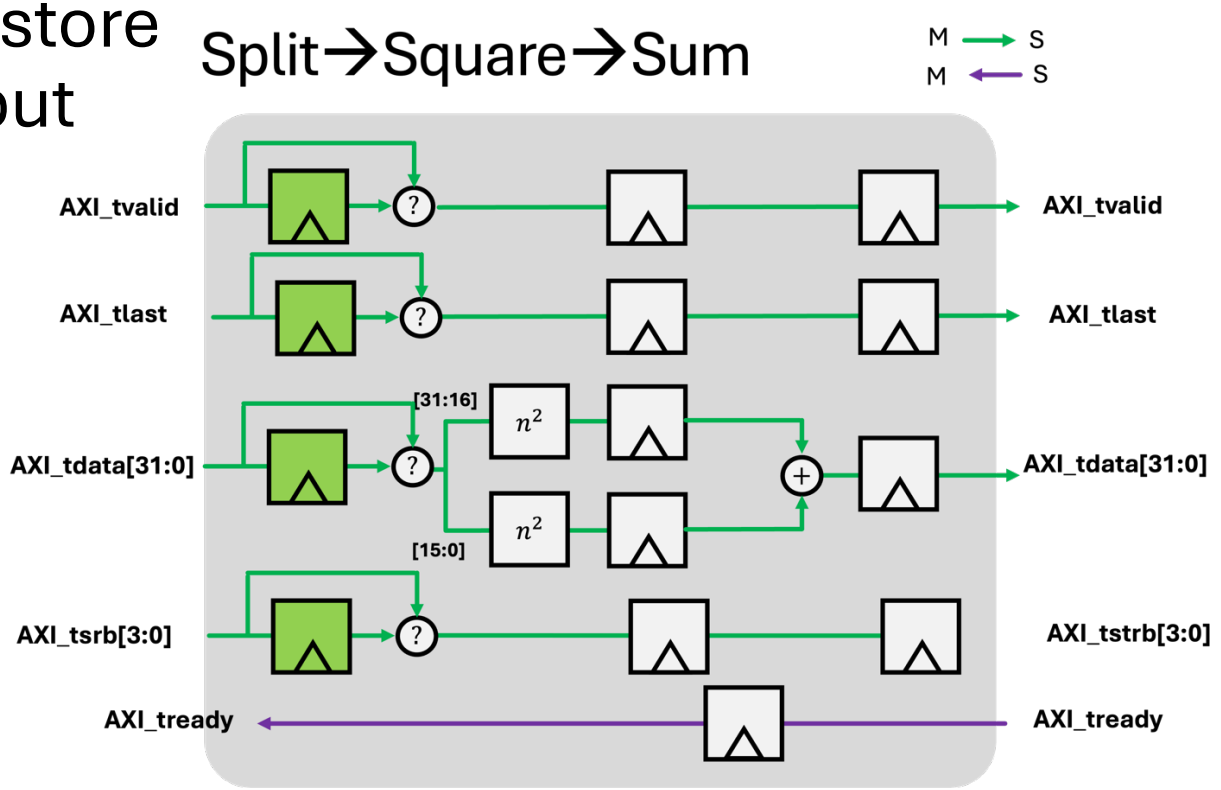
https://ptolemy.berkeley.edu/projects/embedded/research/hsc/class.F02/ee249/lectures/lipClass.pdf

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

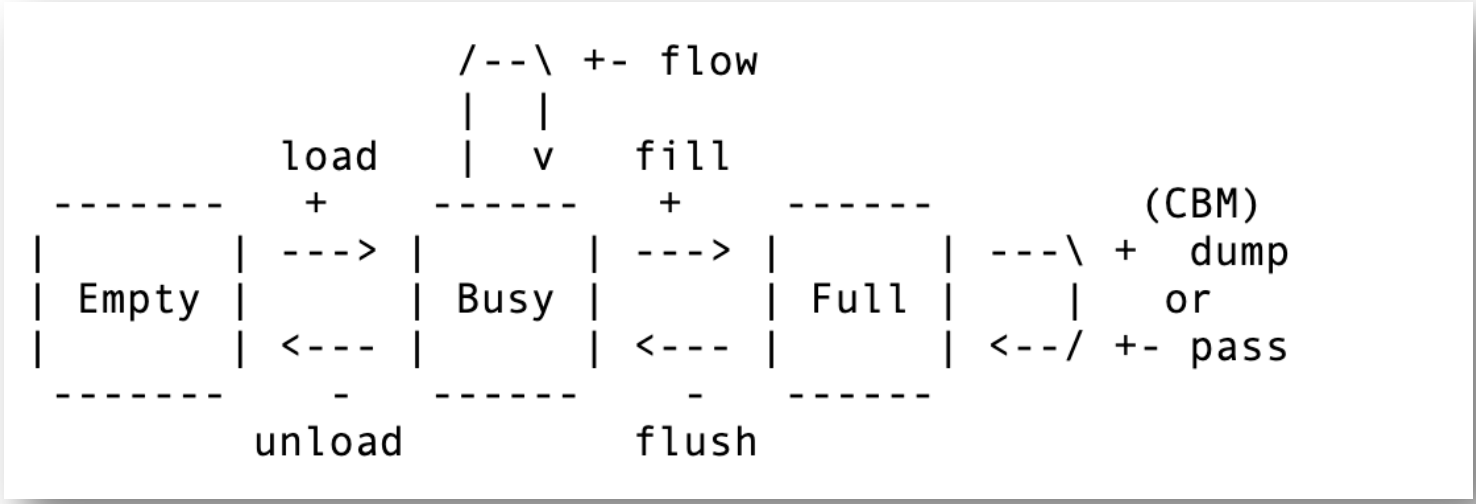
- Kind of an old-school FPGA writeup of a skid buffer found here:

  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.

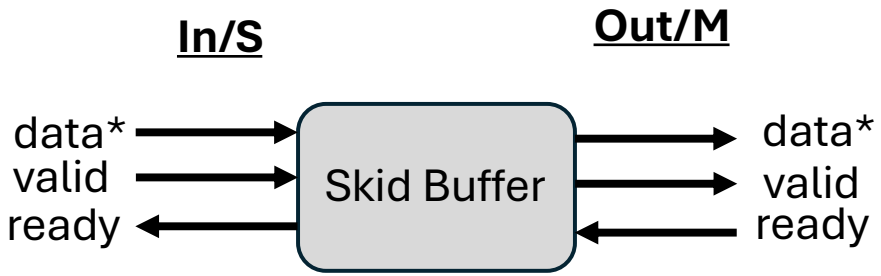
Simple FSM logic

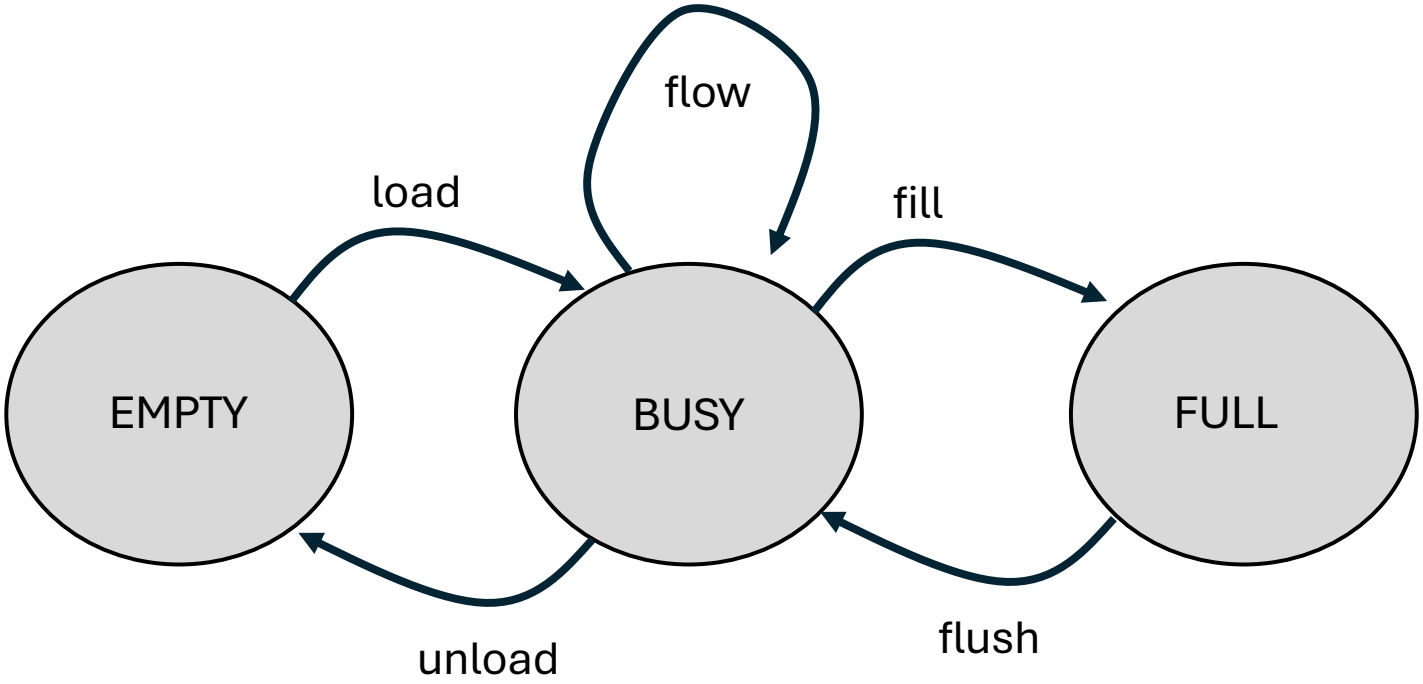
- Three-state FSM can take care of this



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

Example: Skid Buffer





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.*

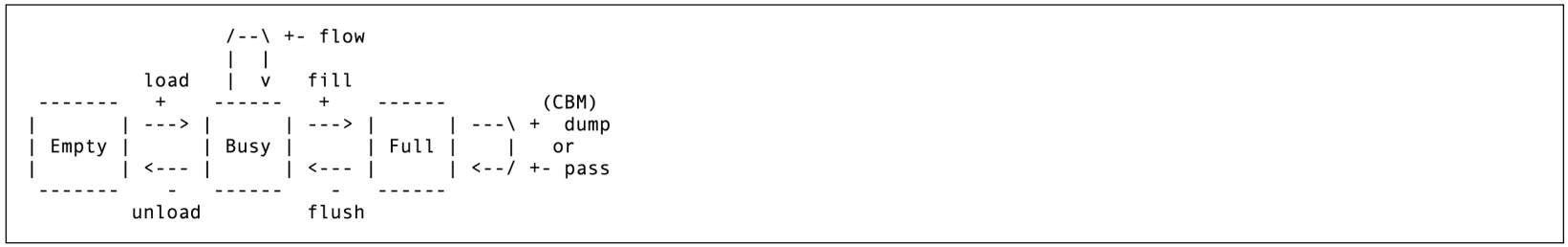
To operate our datapath as a skid buffer, we need to understand which states we want to allow it to be in, and which state transitions we also allow. This skid buffer has three states:

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:

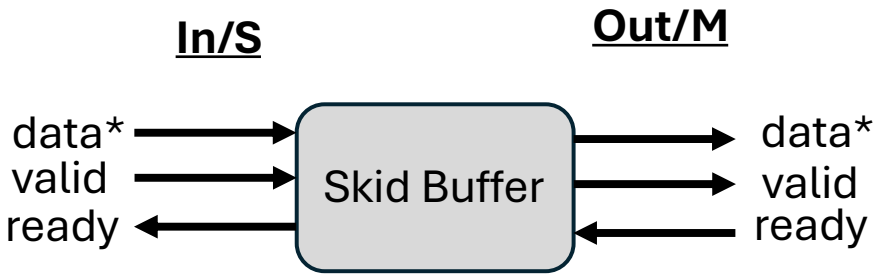
- the input interface inserting a data item into the datapath (`+`)
- the output interface removing a data item from the datapath (`-`)
- both interfaces inserting and removing at the same time (`+-`)

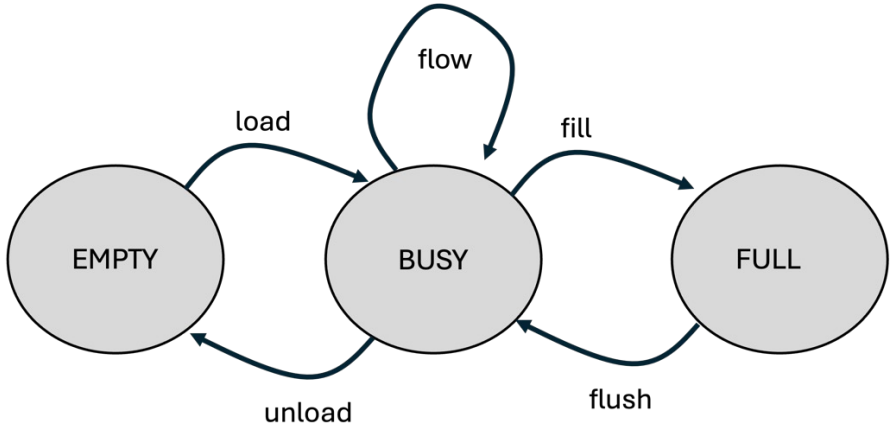
We also descriptively name each transition between states. These names will show up later in the code.



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

Skid Buffer



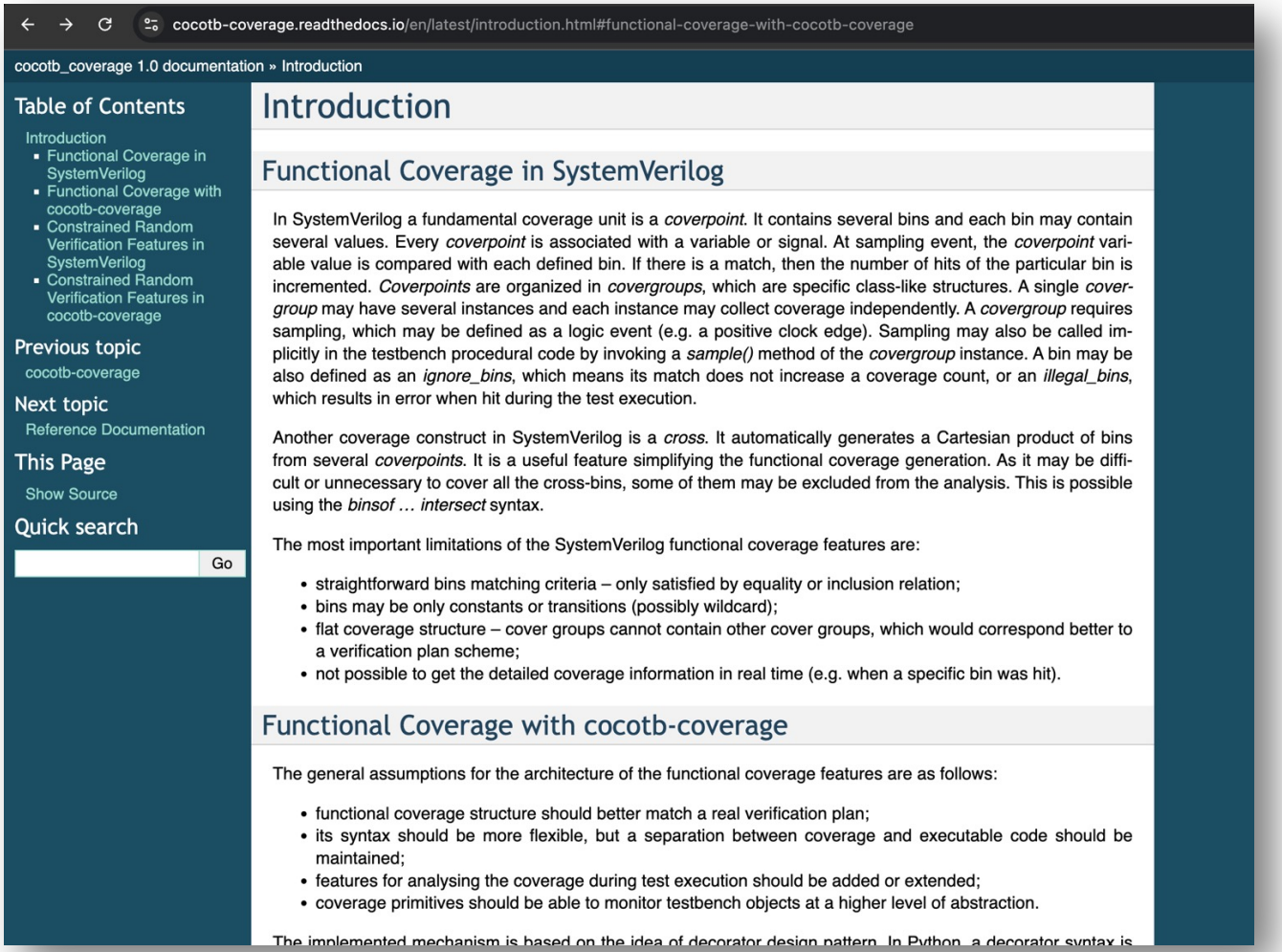


- BUSY is normal operation where data is coming in and out.
- If there's a hiccup on the output side, go to FULL and stall pipeline (s00_tready -> 0)
- If there's a hiccup on the input side, go to EMPTY and stall pipeline (m00_tvalid -> 0)

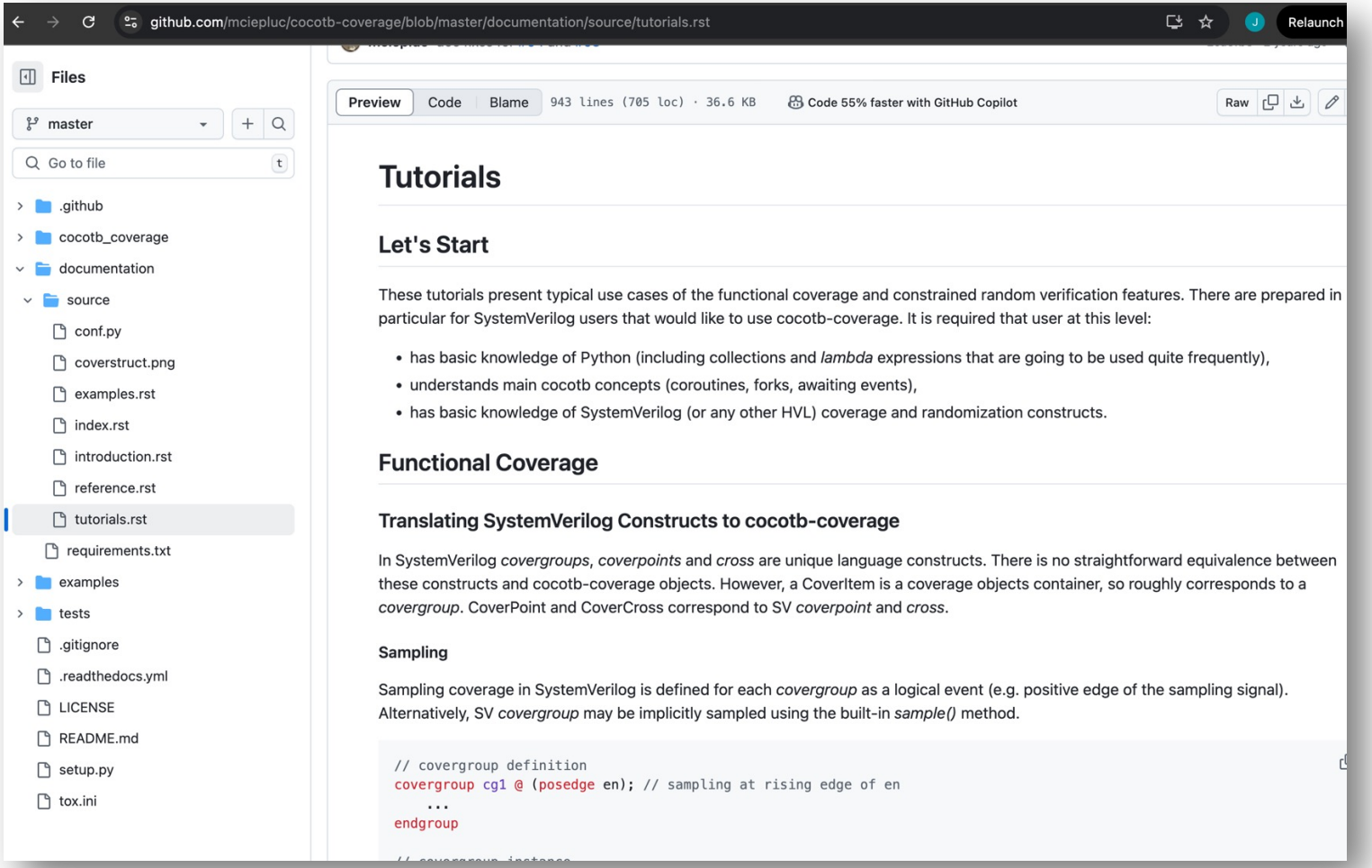
Skid Buffer

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.

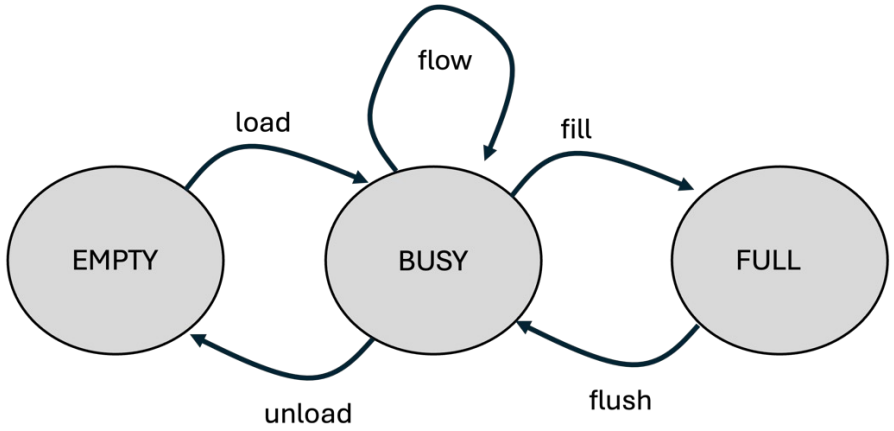
Cocotb Coverage



Another library with ok docs and source code



Cocotb_coverage




```
from cocotb_coverage.coverage import CoverCross, CoverPoint, coverage_db, coverage_section
import constraint
```

- 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

```
SC = coverage_section (
CoverPoint("top.st.state",
            xf=lambda s, ns: s,
            bins=['EMPTY', 'BUSY', 'FULL']
            ),
CoverPoint("top.st.next_state",
            xf=lambda s, ns: ns,
            bins=['EMPTY', 'BUSY', 'FULL']
            ),
CoverCross("top.st.state.cross",
            items=["top.st.state", "top.st.next_state"],

            )
)
```

CoverPoint

```
SC = coverage_section (
CoverPoint("top.st.state",
            xf=lambda s, ns: s,
            bins=['EMPTY', 'BUSY', 'FULL']
            ),
CoverPoint("top.st.next_state",
            xf=lambda s, ns: ns,
            bins=['EMPTY', 'BUSY', 'FULL']
            ),
CoverCross("top.st.state.cross",
            items=["top.st.state", "top.st.next_state"],

            )
)
```

- 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

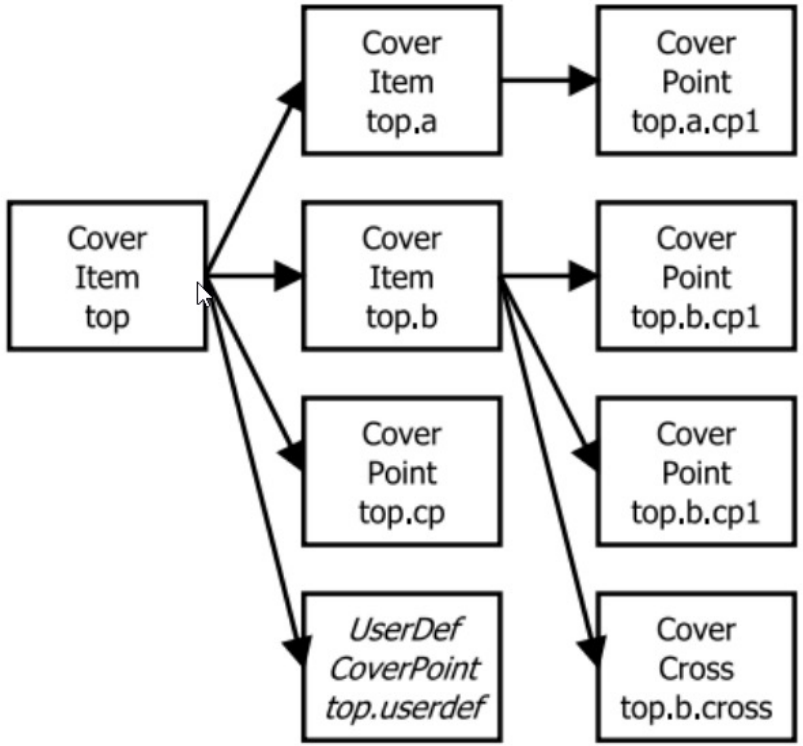
```
SC = coverage_section (
CoverPoint("top.st.state",
            xf=lambda s, ns: s,
            bins=['EMPTY', 'BUSY', 'FULL']
            ),
CoverPoint("top.st.next_state",
            xf=lambda s, ns: ns,
            bins=['EMPTY', 'BUSY', 'FULL']
            ),
CoverCross("top.st.state.cross",
            items=["top.st.state", "top.st.next_state"],

            )
)
```

- 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:'EMPTY', 1:'BUSY', 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:{incoun} 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

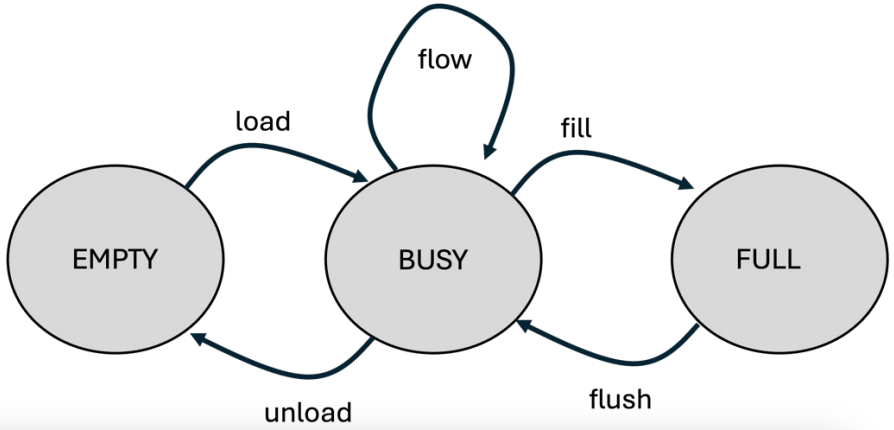
```
top : <cocotb_coverage.coverage.CoverItem object at 0x1029911e0>, coverage=13, size=15
   top.st : <cocotb_coverage.coverage.CoverItem object at 0x10213d3c0>, coverage=13, size=15
      top.st.next_state : <cocotb_coverage.coverage.CoverPoint object at 0x102991390>, coverage=3, size=3
         BIN 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 ('EMPTY', 'FULL') : 0
            BIN ('BUSY', 'EMPTY') : 47
            BIN ('BUSY', 'BUSY') : 103
            BIN ('BUSY', '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) **
**************************************************************************************
** test_skid_buffer.test_a       PASS        6720.00           0.06     104844.35  **
**************************************************************************************
** TESTS=1 PASS=1 FAIL=0 SKIP=0              6720.00           0.11      63130.17  **
**************************************************************************************
```

Or if you prefer pretty to read xml

- I guess

But like… is this good/bad?

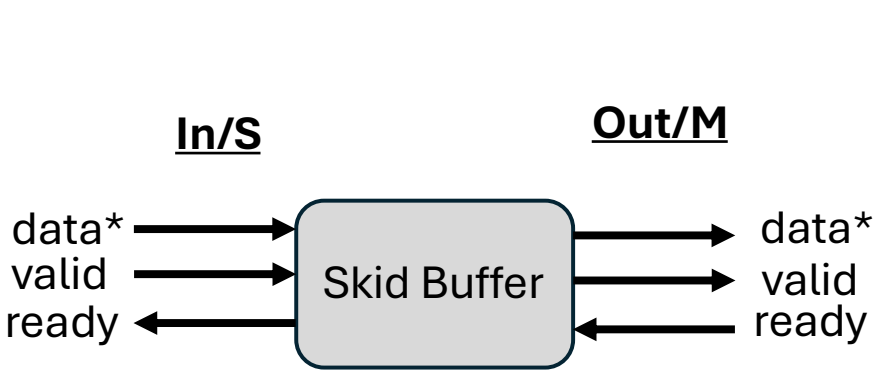
- Anything stand out?

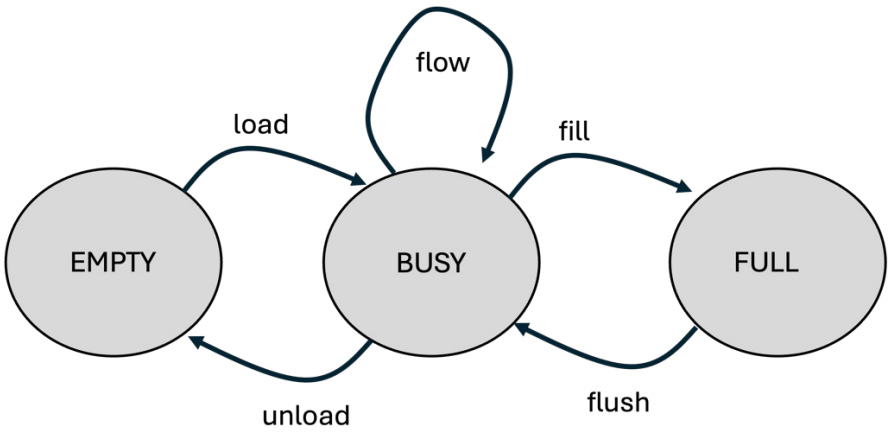



```
top : <cocotb_coverage.coverage.CoverItem object at 0x1029911e0>, coverage=13, size=15
   top.st : <cocotb_coverage.coverage.CoverItem object at 0x10213d3c0>, coverage=13, size=15
      top.st.next_state : <cocotb_coverage.coverage.CoverPoint object at 0x102991390>, coverage=3, size=3
         BIN 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 ('EMPTY', 'FULL') : 0
            BIN ('BUSY', 'EMPTY') : 47
            BIN ('BUSY', 'BUSY') : 103
            BIN ('BUSY', 'FULL') : 2
            BIN ('FULL', 'EMPTY') : 0
            BIN ('FULL', 'BUSY') : 2
            BIN ('FULL', 'FULL') : 305
test_a passed
```

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=['EMPTY', 'BUSY', '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....

```
m/test_skid_buffer.py:143: DeprecationWarning: Use `bv.integer` instead.

   top : <cocotb_coverage.coverage.CoverItem object at 0x10711d270>, coverage=30, size=74
      top.st : <cocotb_coverage.coverage.CoverItem object at 0x1068c80a0>, coverage=13, size=15
         top.st.next_state : <cocotb_coverage.coverage.CoverPoint object at 0x10711d420>, coverage=3, size=3
            BIN EMPTY : 212
            BIN BUSY : 152
            BIN FULL : 307
         top.st.state : <cocotb_coverage.coverage.CoverPoint object at 0x1068c8070>, coverage=12, size=12
            BIN EMPTY : 212
            BIN BUSY : 152
            BIN FULL : 307
            top.st.state.cross : <cocotb_coverage.coverage.CoverCross object at 0x1068c8040>, coverage=7, size=9
               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
               BIN ('FULL', 'BUSY') : 2
               BIN ('FULL', 'FULL') : 305
      top.st_sig : <cocotb_coverage.coverage.CoverItem object at 0x10711dab0>, coverage=17, size=59
         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
```

From before

New...

Very Limited Coverage (12.5%)

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.

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 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)
```

Now?

- Better than before for sure
- 22%

Of course also keep making sure it passes the scoreboard checks

```
top.st_sig : <cocotb_coverage.coverage.CoverItem object at 0x1049e1b10>, coverage=22, size=59
   top.st_sig.cross : <cocotb_coverage.coverage.CoverCross object at 0x1049e21d0>, coverage=11, size=48
      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
      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
      BIN ('EMPTY', False, False, False, False) : 0
      BIN ('BUSY', True, True, True, True) : 83
      BIN ('BUSY', True, True, True, False) : 22
      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, 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
      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) : 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
```

```
cocotb.regression                test_a passed
cocotb.regression                **************************************************************************************
                                 ** TEST                          STATUS  SIM TIME (ns)  REAL TIME (s)  RATIO (ns/s) **
                                 **************************************************************************************
                                 ** test_skid_buffer.test_a        PASS       13030.00           0.17      74699.18  **
                                 **************************************************************************************
                                 ** TESTS=1 PASS=1 FAIL=0 SKIP=0              13030.00           0.22      59824.71  **
                                 **************************************************************************************
```

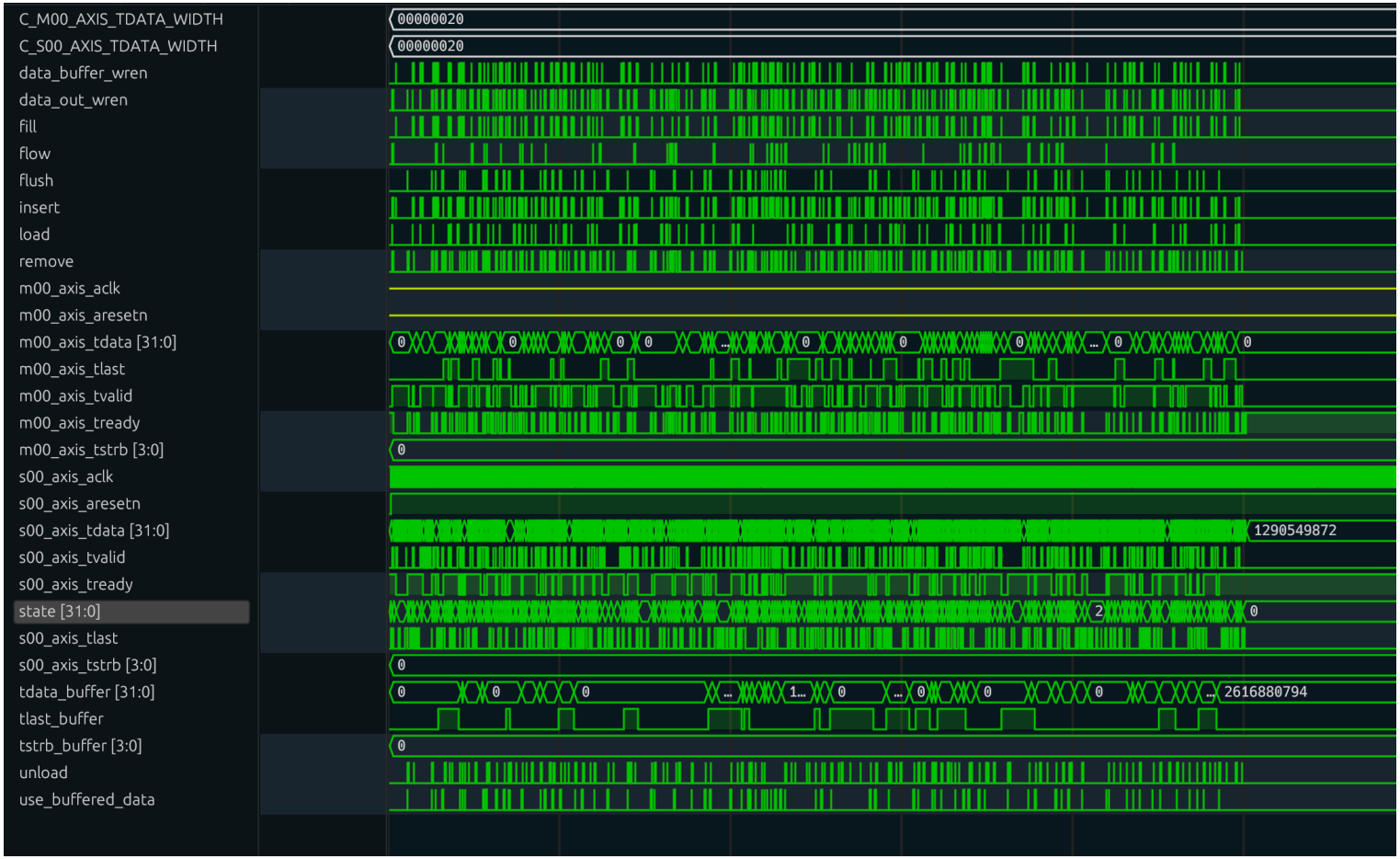
Ignore the Driver

- Just throw crap at this system

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

```python
@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



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

```
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 ('EMPTY', True, True, True, True) : 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 ('EMPTY', 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
```

```
        BIN FULL : 304
test_a passed
**********************************************************************************************
** TEST                          STATUS  SIM TIME (ns)  REAL TIME (s)  RATIO (ns/s) **
**********************************************************************************************
** test_skid_buffer.test_a        PASS       13020.00           0.18      72065.18  **
**********************************************************************************************
** TESTS=1 PASS=1 FAIL=0 SKIP=0              13020.00           0.22      58280.34  **
**********************************************************************************************
```

Run it more?

- Try to catch some other things?

```
##data = {'type':'burst', "contents":{"data": np.array(20*[0]+[1]+30*[0
#data = {'type':'burst', "contents":{"data": np.array(list(range(100)))
#tester.input_driver.append(data)
for x in range(10000):
  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)
```

```
top.st_sig : <cocotb_coverage.coverage.CoverItem object at 0x106ce9b40>, coverage=23, size=59
  top.st_sig.cross : <cocotb_coverage.coverage.CoverCross object at 0x106cea200>, coverage=12, size=48
    BIN ('EMPTY', True, True, True, True) : 0
    BIN ('EMPTY', True, True, True, False) : 0
    BIN ('EMPTY', True, True, False, True) : 178
    BIN ('EMPTY', True, True, False, False) : 541
    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) : 854
    BIN ('EMPTY', False, True, False, False) : 1645
    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) : 233
    BIN ('BUSY', True, True, True, False) : 761
    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 ('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) : 571
    BIN ('FULL', False, False, True, False) : 1711
    BIN ('FULL', False, False, False, True) : 0
    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