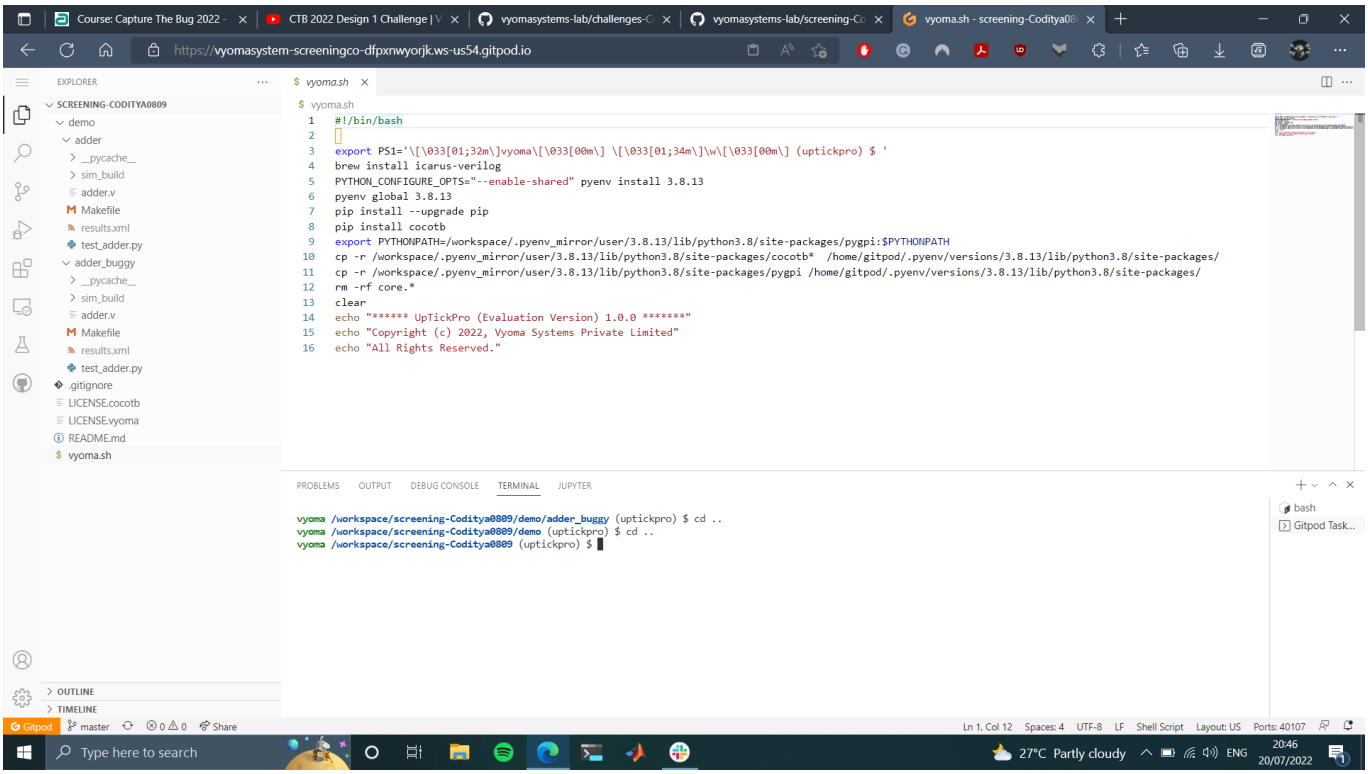
Level-1 Design-1 Verification

The verification environment was setup using Vyoma's UpTickPro provided for the hackathon.



Verification Environment

The CoCoTb based Python test was developed as explained. The test drives inputs to the Design Under Test (31-to-1 multiplexer module here) which takes in 31 2-bit inputs *inp0* - *inp30* and gives 2-bit output *out* based on the 5-bit select line *sel*.

Test 1 : Input 12 not propagating to the output

The values were assigned to the input port using

```
for i in range (31):
    if i != 12 and i != 13:
        dut._id("inp"+str(i), extended=False).value = random.randint(0,3)

# giving dut.inp12 a non-zero value not equal to inp13 to detect error always
dut.inp12.value = seeded_rand_value + 2 if (0 == seeded_rand_value) else
seeded_rand_value
dut.inp13.value = (1 + seeded_rand_value) % 4

dut.sel.value = 12
```

The assert statement was used for comparing the mux's output to the expected value.

The following assert statement was used:

```
assert dut.out.value == dut.inp12.value, f"Mux output is incorrect: Expected
Output != Model Output : {dut.inp12.value} != {dut.out.value}. Expected Value:
{dut.inp12.value}"
```

Test Scenario **(Important)**

- Test Inputs: Inp0 - Inp31 = random values between 0 and 3, both inclusive, Inp12 was reluctantly chosen to be different from Inp13 to detect the bug. Inp12 was also chosen to be non-zero.
- Expected Output: Inp12 value = 2'b10 = 2 in decimal
- Observed Output in the DUT dut.out.value = Inp13.value = 2'b00 = 0 in decimal

Test 2 : Input 30 not propagating to the output

The values were assigned to the input port using

```
for i in range (31):
    dut._id("inp"+str(i), extended=False).value = random.randint(0,3)

dut.sel.value = 30
```

The assert statement was used for comparing the mux's output to the expected value.

The following assert statement was used:

```
assert dut.out.value == dut.inp30.value, f"Mux output is incorrect: Expected
Output != Model Output : {dut.inp30.value} != {dut.out.value}. Expected Value:
{dut.inp30.value}"
```

Test Scenario **(Important)**

- Test Inputs: Inp0 - Inp30 = random values between 0 and 3, both inclusive
- Expected Output: Inp30 value = 2'b11 = 3 in decimal
- Observed Output in the DUT dut.out.value = 2'b00 = 0 in decimal

Failed Test Cases

```
vyoma /workspace/challenges-Coditya0809/level1_design1 (uptickpro) $ make
make -f Makefile results.xml
make[1]: Entering directory '/workspace/challenges-Coditya0809/level1_design1'
MODULE=test_mux    TESTCASE= TOPLEVEL=mux         TOPLEVEL_LANG=verilog \
         /home/linuxbrew/.linuxbrew/bin/vvp -M /workspace/.pyenv_mirror/user/3.8.13/lib/python3.8/site-packages/cocotb/libs -m libcocotbvpi_icarus   sim_build/sim.vvp
     -.--ns INFO     cocotb.gpi                         ..mbed/gpi_embed.cpp:76   in set_program_name_in_venv        Did not detect Python virtual environment. Using sy
stem-wide Python interpreter
     -.--ns INFO     cocotb.gpi                         ../gpi/GpiCommon.cpp:99   in gpi_print_registered_impl       VPI registered
     0.00ns INFO     Running on Icarus Verilog version 11.0 (stable)
     0.00ns INFO     Running tests with cocotb v1.6.2 from /workspace/.pyenv_mirror/fakeroot/versions/3.8.13/lib/python3.8/site-packages/cocotb
     0.00ns INFO     Seeding Python random module with 1658583435
     0.00ns WARNING  Pytest not found, assertion rewriting will not occur
     0.00ns INFO     Found test test_mux.test_mux_duplicate_select
     0.00ns INFO     Found test test_mux.test_mux_last_input
     0.00ns INFO     running test_mux_duplicate_select (1/2)
     0.00ns INFO     ##### CTB: Develop your test here ########
     2.00ns INFO     test_mux_duplicate_select failed
                     Traceback (most recent call last):
                       File "/workspace/challenges-Coditya0809/level1_design1/test_mux.py", line 27, in test_mux_duplicate_select
                         assert dut.out.value == dut.inp12.value, f"Mux output is incorrect: Expected Output != Model Output : {dut.inp12.value} != {dut.out.value}. Exp
ected Value: {dut.inp12.value}"
                     AssertionError: Mux output is incorrect: Expected Output != Model Output : 10 != 00. Expected Value: 10
     2.00ns INFO     running test_mux_last_input (2/2)
     4.00ns INFO     test_mux_last_input failed
                     Traceback (most recent call last):
                       File "/workspace/challenges-Coditya0809/level1_design1/test_mux.py", line 42, in test_mux_last_input
                         assert dut.out.value == dut.inp30.value, f"Mux output is incorrect: Expected Output != Model Output : {dut.inp30.value} != {dut.out.value}. Exp
ected Value: {dut.inp30.value}"
                     AssertionError: Mux output is incorrect: Expected Output != Model Output : 11 != 00. Expected Value: 11
     4.00ns INFO     ********************************************************************************************
                     ** TEST                                STATUS  SIM TIME (ns)  REAL TIME (s)  RATIO (ns/s) **
                     ********************************************************************************************
                     ** test_mux.test_mux_duplicate_select   FAIL          2.00           0.00       714.16  **
                     ** test_mux.test_mux_last_input         FAIL          2.00           0.00      1886.45  **
                     ********************************************************************************************
                     ** TESTS=2 PASS=0 FAIL=2 SKIP=0                     4.00           0.01       273.56  **
                     ********************************************************************************************

make[1]: Leaving directory '/workspace/challenges-Coditya0809/level1_design1'
vyoma /workspace/challenges-Coditya0809/level1_design1 (uptickpro) $
```

Output mismatches for the above inputs proving that there is a design bug

Design Bug

Based on the above test inputs and analysing the design, we see the following

```
5'b01010: out = inp10;
5'b01011: out = inp11;
5'b01101: out = inp12;    <==== BUG
5'b01101: out = inp13;
5'b01110: out = inp14;
5'b01111: out = inp15;
...
5'b11101: out = inp29;    <==== BUG
default: out = 0;
```

For the mux design, the logic for case 5'b01101 is repeated twice. Therefore, only the first case: Inp12 was being executed and the loop was breaking. Therefore instead of `5'b01101` the case for inp12 must be `5'b01100`. The case for propagating Inp30 was also written as `5'b11110: out = inp30`

Design Fix

Updating the design and re-running the test makes the test pass.

```
vyoma /workspace/challenges-Coditya0809/level1_design1 (uptickpro) $ make
make -f Makefile results.xml
make[1]: Entering directory '/workspace/challenges-Coditya0809/level1_design1'
/home/linuxbrew/.linuxbrew/bin/iverilog -o sim_build/sim.vvp -D COCOTB_SIM=1 -s mux          -f sim_build/cmds.f -g2012   /workspace/challenges-Coditya0809/level1_desi
gn1/mux.v
MODULE=test_mux      TESTCASE= TOPLEVEL=mux           TOPLEVEL_LANG=verilog \
         /home/linuxbrew/.linuxbrew/bin/vvp -M /workspace/.pyenv_mirror/user/3.8.13/lib/python3.8/site-packages/cocotb/libs -m libcocotbvpi_icarus   sim_build/sim.vvp
     -.--ns INFO     cocotb.gpi                        ..mbed/gpi_embed.cpp:76   in set_program_name_in_venv        Did not detect Python virtual environment. Using sy
stem-wide Python interpreter
     -.--ns INFO     cocotb.gpi                        ../gpi/GpiCommon.cpp:99   in gpi_print_registered_impl       VPI registered
     0.00ns INFO     Running on Icarus Verilog version 11.0 (stable)
     0.00ns INFO     Running tests with cocotb v1.6.2 from /workspace/.pyenv_mirror/fakeroot/versions/3.8.13/lib/python3.8/site-packages/cocotb
     0.00ns INFO     Seeding Python random module with 1658583663
     0.00ns WARNING  Pytest not found, assertion rewriting will not occur
     0.00ns INFO     Found test test_mux.test_mux_duplicate_select
     0.00ns INFO     Found test test_mux.test_mux_last_input
     0.00ns INFO     running test_mux_duplicate_select (1/2)
     0.00ns INFO     ##### CTB: Develop your test here ########
     2.00ns INFO     test_mux_duplicate_select passed
     2.00ns INFO     running test_mux_last_input (2/2)
     4.00ns INFO     test_mux_last_input passed
     4.00ns INFO     ****************************************************************************************
                     ** TEST                                STATUS  SIM TIME (ns)  REAL TIME (s)  RATIO (ns/s) **
                     ****************************************************************************************
                     ** test_mux.test_mux_duplicate_select   PASS           2.00           0.00        548.76  **
                     ** test_mux.test_mux_last_input         PASS           2.00           0.00       2593.57  **
                     ****************************************************************************************
                     ** TESTS=2 PASS=2 FAIL=0 SKIP=0                      4.00           0.01        288.34  **
                     ****************************************************************************************

make[1]: Leaving directory '/workspace/challenges-Coditya0809/level1_design1'
vyoma /workspace/challenges-Coditya0809/level1_design1 (uptickpro) $
```

The updated design is checked in as mux_fix.v

Verification Strategy

Owing to the simplicity of the design, the verification stratergy I followed was to check the design file very carefully and to my delight, I found that there was a duplicate case statement near Inp11 and Inp12 output assignments. This might have further resulted in a corner case of Inp30 being missed out. Once I found out these bugs, I have written a python module using Cocotb to test this mux design for bugs.

Is the verification complete ?

Yes! The verification for this design is complete because all the cases have been verified manually in the design and there is no chance of any other value other than the selected line to propagate to the output.

We could ensure coverage and achieve coverage closure for this design since it was a simple combinational logic and had manageable input values. I am very aware that as the designs get complicated, the coverage closure becomes very very tricky and also becomes a tireless job for the Verification engineer.