COSE221: Computer Architecture
Design Lab #2

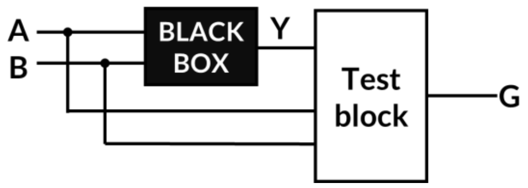
Due: May 15, 2023 (Monday) 11:59pm on Blackboard

Total score: 20 pts

In this lab, you will practice implementing *simple* combinational blocks using SystemVerilog. Since you learned to use CAD tools (i.e. Icarus Verilog and GTKWave) for SystemVerilog coding, you are prepared for create your own SystemVerilog modules. It's time to dive into the practice of digital system design!!

1. Module description

Let us assume you are requested to design a digital block which can verify the logical operations of a BLACKBOX gate. The output signal G (meeting 'GOOD!') becomes '1' if the BLACKBOX gate works correctly. Note that the test block includes three binary input signals (i.e. A, B, and Y) and one output signal (i.e. G).



2. Combinational logic design based on a Boolean equation

The following Karnaugh map represents the logical behavior of the test block. The optimized Boolean equation is also provided below. Implement the test block module using the SystemVerilog logic operators. Note that you need to use only `assign` statements.

Y \ AB	00	01	11	10
0	0	0	1	0
1	1	1	0	1

$$G = \bar{A}Y + \bar{B}Y + AB\bar{Y}$$

The skeleton code of the test block ("`test1.sv`") is provided as shown below. Please complete the provided `test1` module. The testbench ("`tb_test1.sv`") for the test block is provided. Please see how the testbench code is designed.

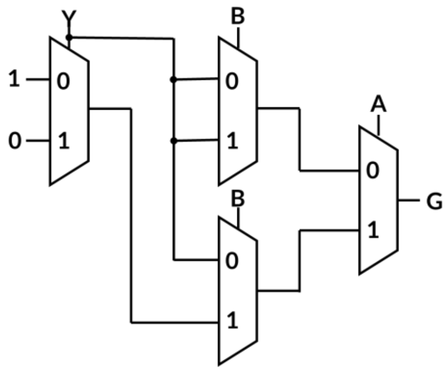
```
module test1(a, b, y, good);
    input        a, b, y;
    output       good;

    /* FILL THIS */
endmodule
```

You can compile and execute the testbench code as follows.

```
$ iverilog -g2005-sv -o test1 tb_test1.sv test1.sv
$ vvp -v test1
```

3. Combinational logic design using multiplexers

The test block module can be implemented using 2:1 multiplexers as shown in the following schematic.



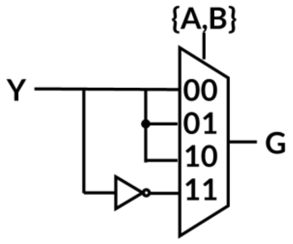
Please complete the provided `test2` module that implements the schematic above. You may declare internal nodes if required. A 2:1 multiplexer needs to be implemented using a ternary operator of SystemVerilog. Note that you need to use only `assign` statements.

You can compile and execute the testbench code as follows.

```
$ iverilog -g2005-sv -o test2 tb_test2.sv test2.sv
$ vvp -v test2
```

4. Combinational logic design using a multiplexer

The test block module can be also implemented using a 4:1 multiplexer as follows.



Please complete the provided `test3` module that implements the schematic above. You may declare internal nodes if required. Note that a 4:1 multiplexer can be implemented using a `case` statement.

You can compile and execute the testbench code as follows.

```
$ iverilog -g2005-sv -o test3 tb_test3.sv test3.sv
$ vvp -v test3
```

5. Structural module design

The test block can be implemented with a structural model by instantiating multiple submodules. Please complete the provided `test4` module that implements the structural model of the `test2` module (i.e. the schematic in Section 3). Note that a 2:1 multiplexer can be implemented using tristate logic. You need to first complete `mux2` module that implements a 2:1 multiplexer using tristate logic, then complete the structural module of `test4`. Please be advised that the output of tristate logic should be a `tri` type signal.

You can compile and execute the testbench code as follows.

```
$ iverilog -g2005-sv -o test4 tb_test4.sv test4.sv
$ vvp -v test4
```

6. What to do

(a) Complete the provided modules (i.e. `test1.sv`, `test2.sv`, `test3.sv`, and `test4.sv`). Simulate each module using the provided testbench. You don't need to modify the testbench modules. Capture the waveform of each design. You need to make the waveforms displayed until 45 ns. Your waveforms should include input and output signals of your design. Embed the captured images in the report document.

(b) Figure out the basic logic gate in the BLACKBOX module that can be verified using the test block. Namely, the output `GOOD` will become '`1`' if the logic gate works correctly. You need to answer the name of the logic gate in the BLACKBOX module in the report document.

(c) Compress your PDF file (report), source codes (`test1.sv`, `test2.sv`, `test3.sv`, and `test4.sv`), the generated VCD files (`*.vcd`), and output files (`*.out`) in **one zip file**. You must name your zip file as "`FirstName_LastName.zip`". (e.g. `Gildong_Hong.zip` for Gildong Hong) If your submission file does not meet this rule, we will reduce 1 point from your score. 🙁

<NOTE>

SystemVerilog syntax `always_comb`, `always_ff`, and `always_latch` is supported by Icarus Verilog v11.0. Unfortunately, the older versions of Icarus Verilog cannot understand `always_comb`, `always_ff`, and `always_latch`. You can install Icarus Verilog v11.0 from Ubuntu's default repository if you are using Ubuntu newer than version 22.04. If not, you can try one of the following methods.

Method 1 (recommended):

Change always statements in the source codes as follows.

```
always_comb → always @ (*)
always_ff → always
```

Method 2:

You can compile the newer version of Icarus Verilog from the source code. Please follow the instructions in https://iverilog.fandom.com/wiki/Installation_Guide#Compiling_on_Linux/Unix. Before installing the newer version of Icarus Verilog, you need to remove the installed Icarus Verilog.

```
$ sudo apt purge iverilog
$ sudo apt update
$ sudo apt install bison, flex, autoconf, gperf
$ mkdir -p workspace
$ cd workspace
$ git clone https://github.com/steveicarus/iverilog.git
$ cd iverilog
$ git checkout --track -b v11-branch origin/v11-branch
$ git pull
$ ./configure
$ make
$ sudo make install
```