**PROBLEM STATEMENT**

This experiment observes how Verilog logic primitives handle inputs of the basic logic types: Logical High (‘1’), Logical Low (‘0’), Undefined (‘X’) and High Impedance (‘Z’). A fundamental multiplexer is constructed from Verilog logic primitives and the output is observed for all cases to extrapolate the behavioral of fundamental set {AND, OR, NOT} primitives during simulation.

**KEYWORDS:** MUX, Logical High, Logical Low, Undefined, High Impedance, Hi-Z, 1, 0, X, Z,AND, OR, NOT not(), and(), or(), Select Line, Control Signal.

**1.1 INTRODUCTION**

The process of multiplexing (MUX) consists of selecting one or more inputs based on a control signal or select line. The simplest multiplexer is one that acts as a single-pole double-throw switch, selecting which of two inputs are passed to the output on the basis of a third control input. In digital design, multiplexing is used extensively to implement conditional logic. A logical expression which can be used to implement MUX logic using the universal set {AND, OR, NOT} is shown as a Boolean expression in Eq. 1.1(1) below:

(1)

From this equation, it is possible to illustrate the behavior of the 2-1 MUX for all values of the determinate binary set comprising Logical High and Logical Low {‘1’,’0’} which can be computed for inputs *A*, *B* and *SEL*. These values can be seen in Table 1.1.

*Table 1.1* Truth Table for 2-Input Multiplexer

|  |  |  |  |
| --- | --- | --- | --- |
| *A* | *B* | *SEL* | OUT |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

It should be noted that this table only considers binary values from the determinate set. Output & input combinations for a the MUX are defined by equation 1.1(2) below. Let n be the number of logical inputs, NB the number of binary values the inputs can assume and NC the resulting total number of combinations.

(2)

For the the multiplexer described by Eq. 1.1(1), a gate level implementation is shown below in Figure 1.1.



Figure 1.1 Gate Schematic for 2-Input MUX

Within the complete set of all binary values the states of undefined (‘X’) and high impedance (‘Z’), or Hi-Z, are also included. The logical state of undefined occurs when the voltage at a given node is different enough in potential from both Logical Low, or ground, and Logical High, or the supply voltage, to be determined. Thus, the result of reading an undefined value could be interpreted as either a 0 or 1 and cannot be read reliably. The Hi-Z state models an open circuit pathway, where the resistance between two the two open terminals is theoretically infinite.

A functioning multiplexer will be able to pass logical values of the set {‘0’, ’1’} in accordance with Eq. 1.1(1) and may exhibit unpredictable behavior when members of the indeterminate set of {‘Z’, ‘X’} are passed to any of the inputs.

**1.2 METHODOLOGY**

To determine the behavior of the fundamental set of Verilog primitives over the complete binary set {‘1’,’0’,’Z’,’X’} the binary output of a logical function must be observed for all possible input combinations. The basic MUX function described by Eq 1.1(1) has two logical inputs, one control line and one logical output. In addition the internal nodes of the Multiplexer, shown as *A1*, *B1* and *SEL\_N* in Fig 1.1 above must be analyzed to determine the behavior of each individual primitive component of the MUX.

To satisfy this collection of data, two Verilog modules are constructed. One module contains the declaration definition of the basic mux using standard Verilog code and the described primitive gates. This module must be able to handle all combinational inputs of the binary set and output a single logical value which is controlled by the Select (*SEL*) input. This module can be seen below in section 1.3 as Module 1.1 *2MUX\_1.v*. In addition to the MUX module, a testbench module must also be implemented to observe the *2MUX\_1* module’s output given different input conditions. This module is also detailed in section 1.3 and can be seen as Module 1.2 *tb\_2MUX\_1*.v.

In order to test all input combinations, each variable must be assigned each value of the complete binary set while the others are retain a constant value. This necessitates each of the three variables assume four possible values. The number of input combinations required for this experiment is calculated to below in Eq 1.2(3) where there are three inputs and four binary values.

(3)

This result necessitates 64 total test cases ranging from 000 to ZZZ for inputs *A*, *B*, and *SEL* respectively. An exhaustive test strategy was adopted to sequentially test and observe each input combination starting from 000. In this simulation, each input combination represents a unique test vector.

The testbench and behavioral modules must then be compiled using the Verilog Compile Simulator tool (VCS). If compiled with no warnings or errors, the behavioral simulation will be run and the output recorded. Figure 1.2 shows the captured behavioral waveforms of all recorded experimental values during simulation.

* 1. **MODULE FILES & SIMULATION RESULTS**

*Module 1.1*-- 2MUX\_1.v

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\* \*\*\*

\*\*\* EE 526 L Experiment #1 Kyle E. Keislar, Spring, 2020 \*\*\*

\*\*\* \*\*\*

\*\*\* Familiarization with Linux and the Synopsys VCS Simulator \*\*\*

\*\*\* \*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\* Filename: MUX2\_1.v \*\*\*

\*\*\* Author: Kyle E. Keislar \*\*\*

\*\*\* Date: 01/30/2020 \*\*\*

\*\*\* Version: 1.0 \*\*\*

\*\*\* Revised: \*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\* Module Description: 2 Input Multiplexer with Control Line SEL \*\*\*

\*\*\* \*\*\*

\*\*\* A B SEL OUT \*\*\*

\*\*\* 0 0 0 0 \*\*\*

\*\*\* 0 0 1 0 \*\*\*

\*\*\* 0 1 0 0 \*\*\*

\*\*\* 0 1 1 1 \*\*\*

\*\*\* 1 0 0 1 \*\*\*

\*\*\* 1 0 1 0 \*\*\*

\*\*\* 1 1 0 1 \*\*\*

\*\*\* 1 1 1 1 \*\*\*

\*\*\* \*\*\*

\*\*\* Module Limitations: Combinatorial Inputs \*\*\*

\*\*\* \*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//Timescale

`timescale 1 ns **/** 1 ns

// Module Declaration

**module** MUX2\_1**(**OUT**,** A**,** B**,** SEL**);**

//Port Declarations

**output** OUT**;**

**input** A**,** B**,** SEL**;**

//Internal Signals

**wire** A1**,** B1**,** SEL\_N**;**

//Netlist

**not** **(**SEL\_N**,** SEL**);** //Assign select line

**and** **(**A1**,** A**,** SEL\_N**);** //If A is selected, A1 is A

**and** **(**B1**,** B**,** SEL**);** //If B is selected, B1 is B

**or** **(**OUT**,** A1**,** B1**);** //Output is either A1 selected or B1 selected

//Asynchronous Behavior

//Synchronous Behavior

**endmodule**

*Module 1.2*—tb\_2MUX\_1.v

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\* \*\*\*

\*\*\* EE 526 L Experiment #1 Kyle E. Keislar, Spring, 2020 \*\*\*

\*\*\* \*\*\*

\*\*\* Familiarization with Linux and the Synopsys VCS Simulator \*\*\*

\*\*\* \*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\* Filename: tb\_MUX2\_1.v \*\*\*

\*\*\* Author: Kyle E. Keislar \*\*\*

\*\*\* Date: 01/30/2020 \*\*\*

\*\*\* Version: 1.0 \*\*\*

\*\*\* Revised: \*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\* Test Modules: MUX2\_1.v \*\*\*

\*\*\* Test Strategy: Exhaustive testing strategy; Observe output OUT for \*\*\*

\*\*\* Each combination of all logical values (0,1,X,Z) for \*\*\*

\*\*\* inputs A, B, SEL. \*\*\*

\*\*\* \*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//Timescale

`timescale 1ns **/** 1ns

// Module Instantiation

**module** tb\_MUX2\_1**();**

//reg inputs

**reg** A**,** B**,** SEL**;**

//wire outputs

**wire** OUT**;**

MUX2\_1 UUT**(**OUT**,** A**,** B**,** SEL**);**

//Initial Conditions

**initial**

$monitorb **(**" %d out = %b a=%b b= %b sel =%b"**,** $time**,** OUT**,** A**,** B**,** SEL**);** //Output Module variables during simulation on a new line

//Run-time Testing

**initial** **begin**

$vcdpluson**;** //include waveforms in simulation

A **=** 1**;** B **=** 1**;** SEL **=** 1'bx**;** //initalize A, B, SEL

// Test all logical combinations for 3 inputs ==> 4^3 = 64 Test Vectors

//Test Vectors 1-16 (000-0XX) //A B SEL

**#**10 A **=** 0**;** B **=** 0**;** SEL **=** 0**;** //000

**#**10 SEL **=** 1**;** //001

**#**10 SEL **=** 1'BZ**;** //00Z

**#**10 SEL **=** 1'BX**;** //00X

**#**10 A **=** 0**;** B **=** 1**;** SEL **=** 0**;** //010

**#**10 SEL **=** 1**;** //011

**#**10 SEL **=** 1'BZ**;** //01Z

**#**10 SEL **=** 1'BX**;** //01X

**#**10 A **=** 0**;** B **=** 1'bZ**;** SEL **=** 0**;** //0Z0

**#**10 SEL **=** 1**;** //0Z1

**#**10 SEL **=** 1'BZ**;** //0ZZ

**#**10 SEL **=** 1'BX**;** //0ZX

**#**10 A **=** 0**;** B **=** 1'bX**;** SEL **=** 0**;** //0X0

**#**10 SEL **=** 1**;** //0X1

**#**10 SEL **=** 1'BZ**;** //0XZ

**#**10 SEL **=** 1'BX**;** //0XX

//Test Vectors 17-32 (100-1XX)

**#**10 A **=** 1**;** B **=** 0**;** SEL **=** 0**;** //100

**#**10 SEL **=** 1**;** //101

**#**10 SEL **=** 1'BZ**;** //10Z

**#**10 SEL **=** 1'BX**;** //10X

**#**10 A **=** 1**;** B **=** 1**;** SEL **=** 0**;** //110

**#**10 SEL **=** 1**;** //111

**#**10 SEL **=** 1'BZ**;** //11Z

**#**10 SEL **=** 1'BX**;** //11X

**#**10 A **=** 1**;** B **=** 1'bZ**;** SEL **=** 0**;** //1Z0

**#**10 SEL **=** 1**;** //1Z1

**#**10 SEL **=** 1'BZ**;** //1ZZ

**#**10 SEL **=** 1'BX**;** //1ZX

**#**10 A **=** 1**;** B **=** 1'bX**;** SEL **=** 0**;** //1X0

**#**10 SEL **=** 1**;** //1X1

**#**10 SEL **=** 1'BZ**;** //1XZ

**#**10 SEL **=** 1'BX**;** //1XX

//Test Vectors 33-48 (Z00-ZXX)

**#**10 A **=** 1'BZ**;** B **=** 0**;** SEL **=** 0**;** //Z00

**#**10 SEL **=** 1**;** //Z01

**#**10 SEL **=** 1'BZ**;** //Z0Z

**#**10 SEL **=** 1'BX**;** //Z0X

**#**10 A **=** 1'BZ**;** B **=** 1**;** SEL **=** 0**;** //Z10

**#**10 SEL **=** 1**;** //Z11

**#**10 SEL **=** 1'BZ**;** //Z1Z

**#**10 SEL **=** 1'BX**;** //Z1X

**#**10 A **=** 1'BZ**;** B **=** 1'bZ**;** SEL**=**0**;** //ZZ0

**#**10 SEL **=** 1**;** //ZZ1

**#**10 SEL **=** 1'BZ**;** //ZZZ

**#**10 SEL **=** 1'BX**;** //ZZX

**#**10 A **=** 1'BZ**;** B **=** 1'bX**;** SEL**=**0**;** //ZX0

**#**10 SEL **=** 1**;** //ZX1

**#**10 SEL **=** 1'BZ**;** //ZXZ

**#**10 SEL **=** 1'BX**;** //ZXX

//Test Vectors 49-64 (X00-XXX)

**#**10 A **=** 1'Bx**;** B **=** 0**;** SEL **=** 0**;** //X00

**#**10 SEL **=** 1**;** //X01

**#**10 SEL **=** 1'BZ**;** //X0Z

**#**10 SEL **=** 1'BX**;** //X0X

**#**10 A **=** 1'Bx**;** B **=** 1**;** SEL **=** 0**;** //X10

**#**10 SEL **=** 1**;** //X11

**#**10 SEL **=** 1'BZ**;** //X1Z

**#**10 SEL **=** 1'BX**;** //X1X

**#**10 A **=** 1'Bx**;** B **=** 1'bZ**;** SEL**=**0**;** //XZ0

**#**10 SEL **=** 1**;** //XZ1

**#**10 SEL **=** 1'BZ**;** //XZZ

**#**10 SEL **=** 1'BX**;** //XZX

**#**10 A **=** 1'Bx**;** B **=** 1'bX**;** SEL**=**0**;** //XX0

**#**10 SEL **=** 1**;** //XX1

**#**10 SEL **=** 1'BZ**;** //XXZ

**#**10 SEL **=** 1'BX**;** //XXX

**#**10 $finish**;** //cease simulation after 20ns

**end** //stop tracking values

**endmodule**

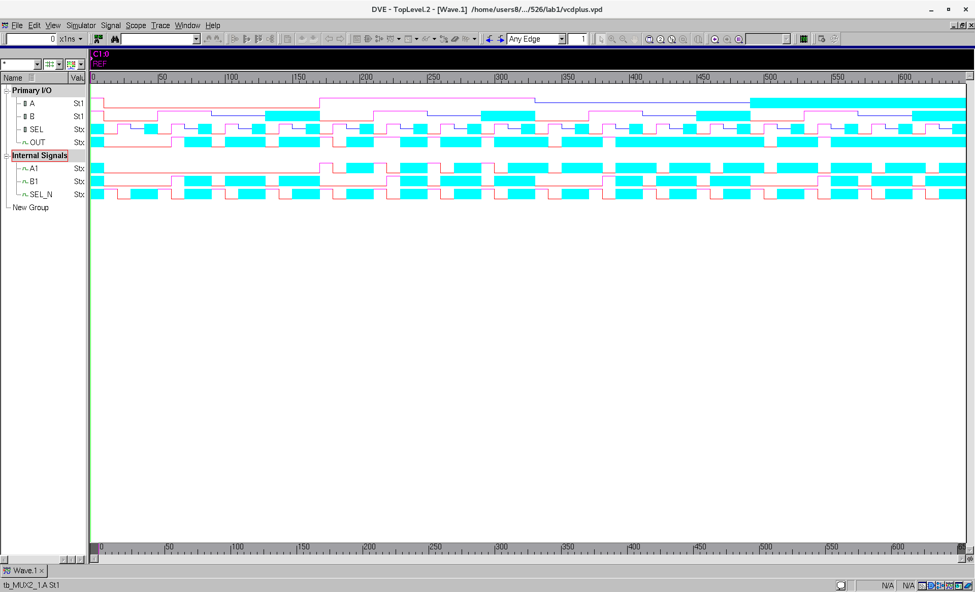


Figure 1.2 Simulation Behavioral Waveforms for Module 1.2

*Table 1.2* Tabulated Experimental Output for 2-1 Input Multiplexer

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Vector #** | **A** | **B** | **SEL** | **OUT** |
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 2 | 0 | 0 | Z | 0 |
| 3 | 0 | 0 | X | 0 |
| 4 | 0 | 1 | 0 | 0 |
| 5 | 0 | 1 | 1 | 1 |
| 6 | 0 | 1 | Z | X |
| 7 | 0 | 1 | X | X |
| 8 | 0 | Z | 0 | 0 |
| 9 | 0 | Z | 1 | X |
| 10 | 0 | Z | Z | X |
| 11 | 0 | Z | X | X |
| 12 | 0 | X | 0 | 0 |
| 13 | 0 | X | 1 | X |
| 14 | 0 | X | Z | X |
| 15 | 0 | X | X | X |
| 16 | 1 | 0 | 0 | 1 |
| 17 | 1 | 0 | 1 | 0 |
| 18 | 1 | 0 | Z | X |
| 19 | 1 | 0 | X | X |
| 20 | 1 | 1 | 0 | 1 |
| 21 | 1 | 1 | 1 | 1 |
| 22 | 1 | 1 | Z | X |
| 23 | 1 | 1 | X | X |
| 24 | 1 | Z | 0 | 1 |
| 25 | 1 | Z | 1 | X |
| 26 | 1 | Z | Z | X |
| 27 | 1 | Z | X | X |
| 28 | 1 | X | 0 | 1 |
| 29 | 1 | X | 1 | X |
| 30 | 1 | X | Z | X |
| 31 | 1 | X | X | X |
| 32 | Z | 0 | 0 | X |
| 33 | Z | 0 | 1 | 0 |
| 34 | Z | 0 | Z | X |
| 35 | Z | 0 | X | X |
| 36 | Z | 1 | 0 | X |
| 37 | Z | 1 | 1 | 1 |
| 38 | Z | 1 | Z | X |
| 39 | Z | 1 | X | X |
| 40 | Z | Z | 0 | X |
| 41 | Z | Z | 1 | X |
| 42 | Z | Z | Z | X |
| 43 | Z | Z | X | X |
| 44 | Z | X | 0 | X |
| 45 | Z | X | 1 | X |
| 46 | Z | X | Z | X |
| 47 | Z | X | X | X |
| 48 | X | 0 | 0 | X |
| 49 | X | 0 | 1 | 0 |
| 50 | X | 0 | Z | X |
| 51 | X | 0 | X | X |
| 52 | X | 1 | 0 | X |
| 53 | X | 1 | 1 | 1 |
| 54 | X | 1 | Z | X |
| 55 | X | 1 | X | X |
| 56 | X | Z | 0 | X |
| 57 | X | Z | 1 | X |
| 58 | X | Z | Z | X |
| 59 | X | Z | X | X |
| 60 | X | X | 0 | X |
| 61 | X | X | 1 | X |
| 62 | X | X | Z | X |
| 63 | X | X | X | X |

* 1. **ANALYSIS**

From the behavioral waveforms shown in Fig 1.2 and their tabulated representations in Table 1.2 multiple conclusions about the functionality of the basic MUX can be derived. A summary of these observations is provided below.

When the inputs are members of the definitive set {‘0’,’1’}, the output is expressed as anticipated and described by Table 1.1. This confirms the defined functionality of the Verilog logic primitives when handling defined input bits.

However when an input is in either undefined or high impedance, the output of the MUX is inconsistent. From table 1.2, the output can be observed to be definitive only if both the SEL control input and the selected input (*A* or *B*) are definitive. The only exception occurs is when *A* and *B* are Logic Low, in this case the output will always be Logic Low regardless of the value of *SEL*. For *SEL* values that are undefined and Hi-Z in all other cases, the output is always undefined. This behavior describes the expected output of attempting to pass an undefined signal. If the select control input cannot be read, neither the value of *A* or *B* can be assumed by the output. At the gate level, this fact is confirmed by the value of node *SEL\_N*, the output of the *SEL* inverter, when *SEL* is either ‘Z’ or ‘X’. In this case *SEL\_N* is always undefined, resulting in the output being undefined.

Undefined inputs do not always necessitate an undefined output. If select is ‘1’, *B* is defined, but the *A* input is undefined or Hi-Z, the MUX will function as if the *A* input was ‘0’ and conform to its known output. This is due to the primitives used to calculate the intermediate signal of *B1*. This observation conforms with the Boolean properties of AND which state that if ‘0’ and any other value are input into an AND function, the resultant output will also be ‘0’. Similarly, if select is ‘0’, the *A* input is defined, but the *B* input is undefined or Hi-Z, the MUX will function as if *B* input was ‘0’.

However, If an undefined or Hi-Z input is selected by the SEL control signal, the corresponding output will be undefined. This result implies that one or both of the internal AND gates of the MUX are passing undefined values to the OR gate. The annulment law of Boolean algebra describes when any value that input to an OR function with 1, to output of the function is the value. Or X + 1 = X. In addition the idempotent law describes the output of an OR function of duplicative inputs to be equal to that the inputs. This means X+X=X. These Boolean principals describe the behavior of OR with non-definitive values. The same principals can be applied when analyzing the AND gates at nodes *A1* and *B1*. By the identity law, any value input to an AND function with 1 must equal that value. Or A.1=A. Similarly, A.A also is equivalent to A.

Finally, If both inputs are Undefined or Hi-Z , then the output is always undefined. From the logical principals previously mentioned this behavior of the MUX is also not surprising. Fig 1.2 shows that any time both of the inputs cannot be read, the output is always undefined even if one of them is selected by the control input.

This MUX circuit is entirely combinatorial, and has no associated synchronous behavior beyond a static propagation delay for all gates. A timing analysis was subsequently not performed.

* 1. **CONCLUSION**

From this experiment multiple conclusions about the behavior of the Verilog logic primitives are apparent. The NOT primitive can only define its output if the input is ‘0’ or ‘1’. If the input is ‘Z’ or ‘X’, the output is undefined. The AND primitive can only define its output if one of the inputs is ‘0’ or both are ‘1’. If one input is ‘Z’ or ‘X’ and the other is not Logic low, the output is undefined. Finally the OR primitive can only define its output if both of the inputs are ‘0’ or ‘1’. If either input is ‘Z’ or ‘X’, the output will be undefined.

This experiment illustrated how undefined or “garbage” values can propagate through a system. When these values are cleared by an operation, they can be subverted and induce undesirable or unintelligible outputs. To avoid this case, care must be taken to discover the causes of any unwarranted ‘Z’ or ‘X’ signals in a design and to ensure that the values do not propagate any errors in logic.

Due to its combinatorial nature, the MUX has no way to handle sudden changes in input or manifest any asynchronous change of values. In addition the 2MUX\_1.v module described in Module 1.1 cannot provide definitive output when any 2 inputs are undefined or Hi-Z. The circuit can function as a MUX with one undefined input when that input is not selected.

* 1. **ADDITIONAL QUESTION**

These two multiplexer codes are identical in their functionality but differ in the provided 2MUX\_1 module used in this experiment . Both share a mux2 declaration specifying two MUX inputs, a select control line, an enable and an output. Both also define internal signals though the pg.10 module explicitly does so. Finally both possess identical select logic with the pg.9 module using a compact if, else syntax to check the value of select, choosing the second input Y if it is 0 and X if it is 1.

These modules differ from 2MUX\_1 because they possess an enable to allow the output to enter a high impedance state. 2MUX\_1 has no such module. In addition, the pg. 10 and 9 modules possess a sensitivity list that can asynchronously update their output based on a change in input. 2MUX\_1 has no sensitivity list.