Lab 1 - Generic Nx1 Mux

Group A - Yuta Akiya, Kyle Le, Megan Luong
Prof. Aly
ECE 4304

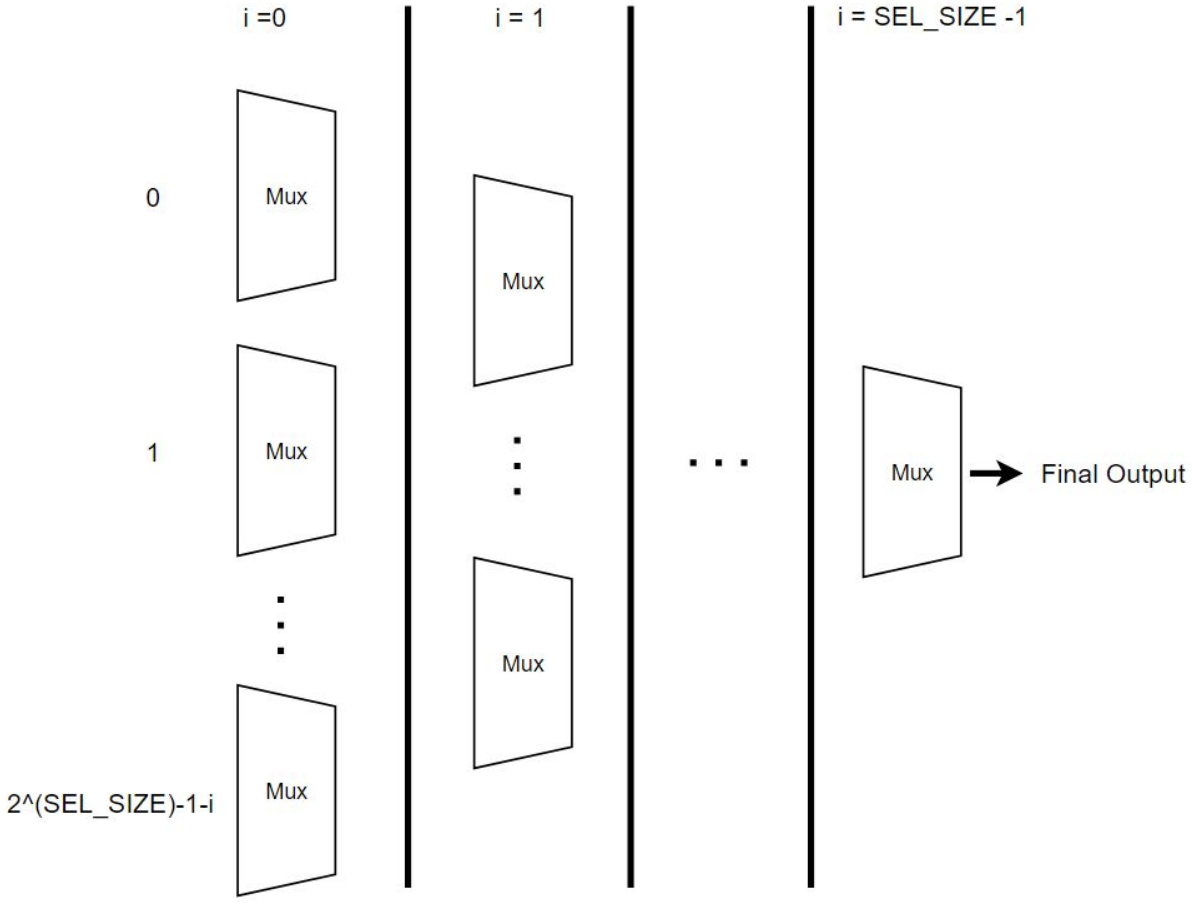
1) Report: Word/pdf report showing the architecture (circuit with details information before you start coding), the trick of the codes, and the list of possible corner cases you cover using testbench, pick the area/resources information from the tool.

Architecture

The architecture of our Nx1 mux consists of layers of 2x1 muxes where the final output is determined by using outputs from a layer as the inputs for the next layer. N, the input size, is defined by the user and the size of the select would be $\log_2(N)$.

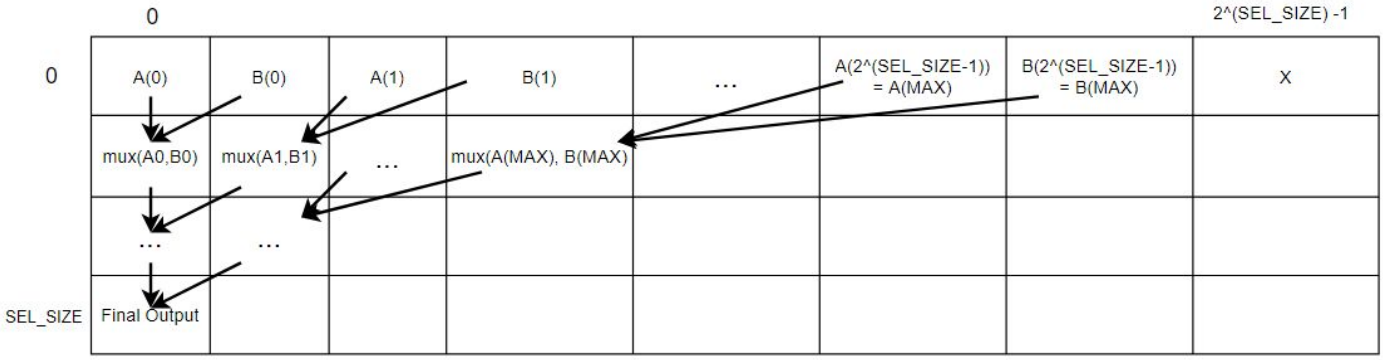
The challenge in making a generic sized mux is parametrization so that the program will work for any input N. For this design, we need to know how many layers of 2x1 muxes are needed as well as how many 2x1 muxes are in each layer with any given input size.

As illustrated in the figure below, the number of muxes was found to be $2^{\text{select size - 1 - i}}$ in the ith layer starting from i = 0. The total number of layers was determined to be select size - 1. Now that the number of layers and the number of 2x1 muxes in each layer is parametrized, the mux size is fully generic and ready to be implemented in VHDL.



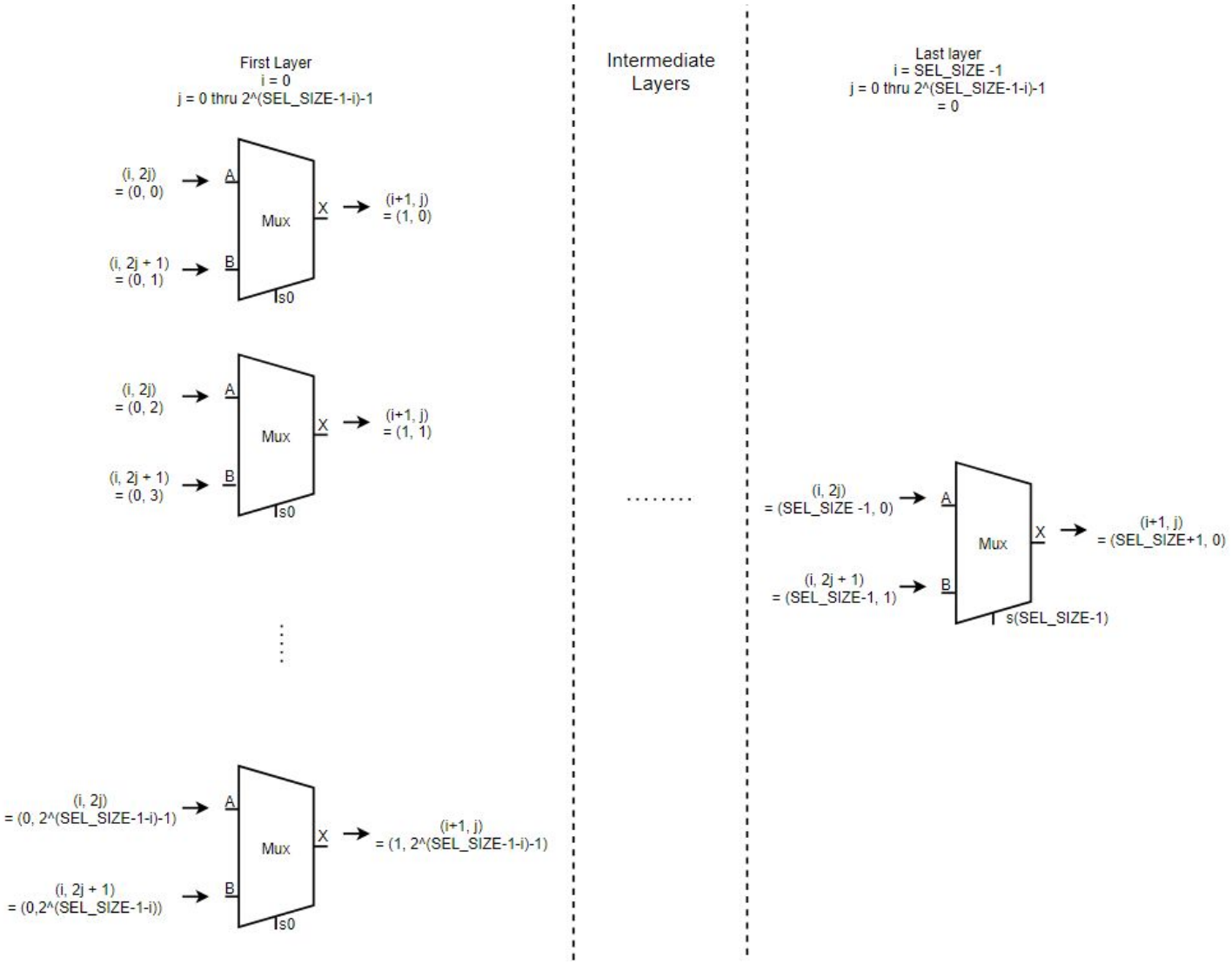
Code Trick/Details

The most challenging part of this lab was finding a method to connect each layer of 2x1 muxes to each other. To do this, a memory mechanism must be designed in order to store the initial inputs and each layer's outputs. A 2-D array of size Select Size + 1 x Input Size is created to serve as the program's memory.



Also, in order for the mux to be a true generic Nx1 mux, with N being any value greater than 1, the code had to be modified to accommodate if it is not a power of 2. To do this, the SEL_SIZE = $\log_2(N)$ was applied the "ceil" (ceiling) value to it. This rounds the result up, allowing for the select to completely cover all N inputs. Since the select can go outside of the N range, the width of the 2-D array is set to 2^{SEL_SIZE} and instantiated with the value of "X". Thus, if the value of the select goes outside the range of the Nx1 mux, it will return an error "X".

The initial inputs are stored in the first row. Then, for each layer, a pair of two adjacent inputs are used as mux inputs and the outputs are stored to the next row. As seen in the figure below, each input is taken from a certain (i, j) address in the 2-D array and stored in another (i, j) address in the next row. This repeating process of taking inputs from the array and storing them back into the array until there is finally one output can be described as a cascading algorithm.

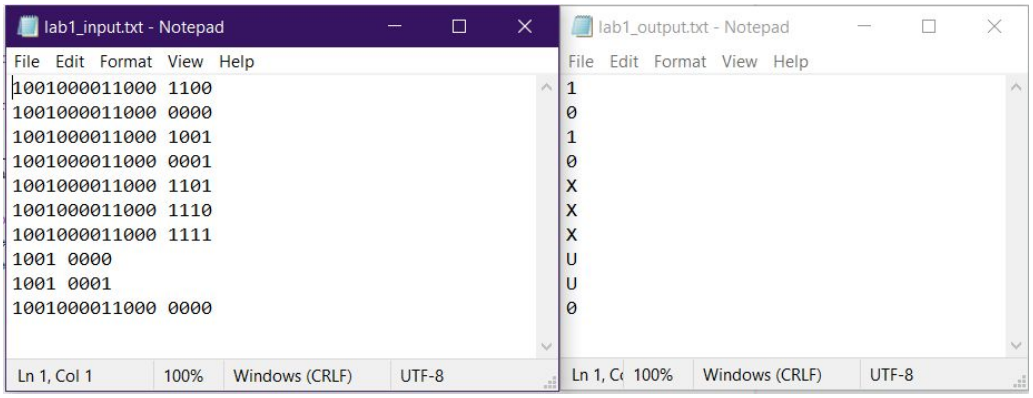

First Layer
i = 0
j = 0 thru 2^(SEL_SIZE-1-i)-1
Intermediate Layers
Last layer
i = SEL_SIZE -1
j = 0 thru 2^(SEL_SIZE-1-i)-1
= 0
(i, 2j)
= (0, 0)
(i, 2j + 1)
= (0, 1)
A
B
Mux
X
s0
(i+1, j)
= (1, 0)
(i, 2j)
= (0, 2)
(i, 2j + 1)
= (0, 3)
(i+1, j)
= (1, 1)
(i, 2j)
= (0, 2^(SEL_SIZE-1-i)-1)
(i, 2j + 1)
= (0,2^(SEL_SIZE-1-i))
(i+1, j)
= (1, 2^(SEL_SIZE-1-i)-1)
(i, 2j)
= (SEL_SIZE -1, 0)
(i, 2j + 1)
= (SEL_SIZE-1, 1)
s(SEL_SIZE-1)
(i+1, j)
= (SEL_SIZE+1, 0)

Corner Cases

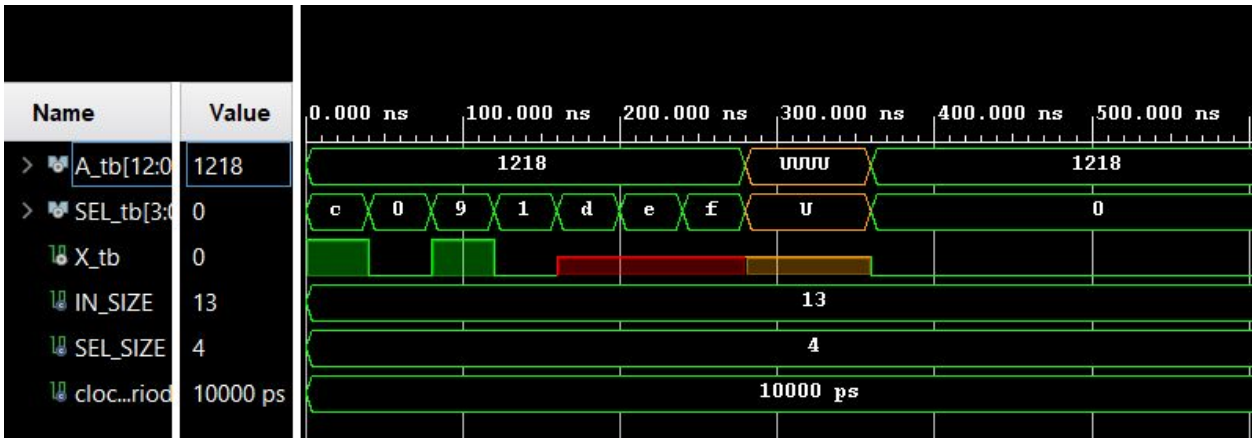
Testbench Algorithm

```
89          while not endfile(file_vectors) loop
90              readline(file_vectors, v_ILINE);
91              read(v_ILINE, v_A);
92              read(v_ILINE, v_space);
93              read(v_ILINE, v_SEL);
94
95              A_tb <= v_A;
96              SEL_tb <= v_SEL;
97
98              wait for 4*clock_period;
99
100             write(v_OLINE, X_tb);
101             writeline(file_results, v_OLINE);
102
103             end loop;
104                 file_close(file_vectors);
105                 file_close(file_results);
106
107             wait;
```

The inputs are read from lab1_input.txt, reading each line for the input variables, v_A, a space, v_space, then the select, v_SEL. The testbench signals then are set to the variable values followed by a 4 clock period wait. The output signal, X_tb, is written to the file for the results, lab1_output.txt. The loop repeats until the end of the file for the inputs. Below are images that display the contents of each text file.



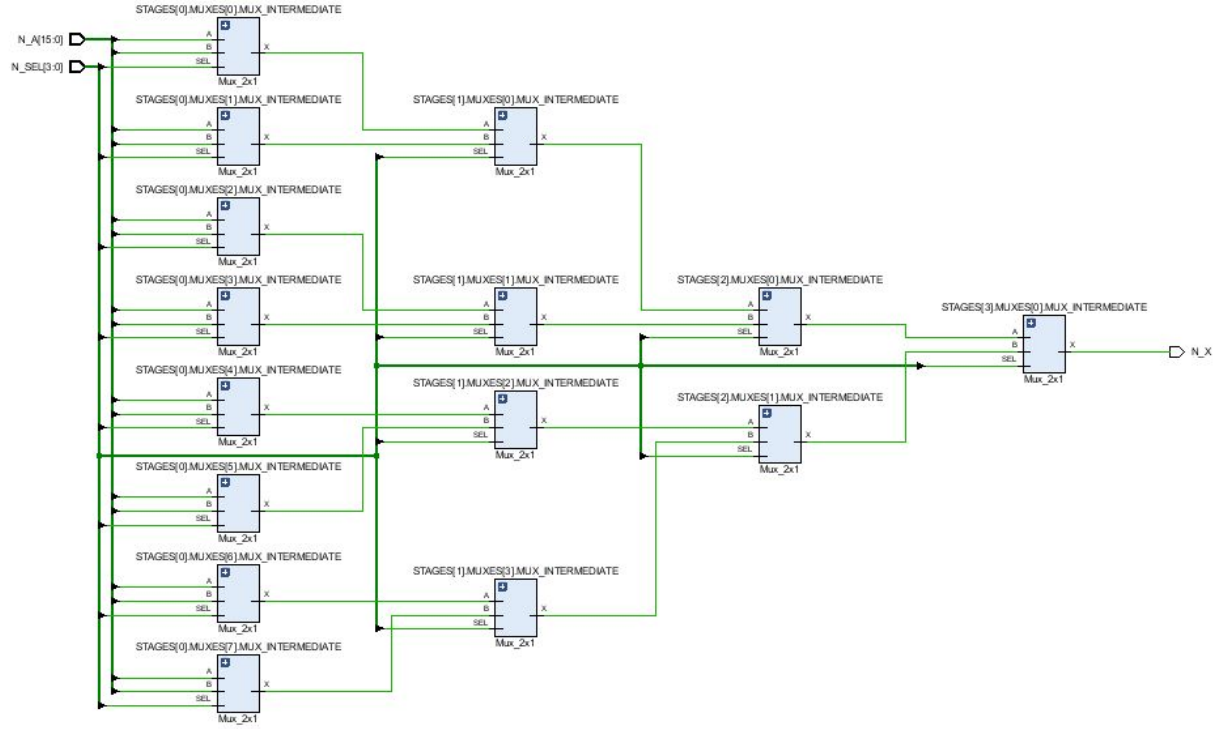
Testbench Results



One of the cases simulated shows a mux with 13 inputs which requires a 4-bit selector. In this case, the selector's maximum is greater than the inputs of the mux. When the select exceeds the inputs of the mux, the output is a don't care term. Another case is attempting to instantiate a number of inputs less than the mux capacity. If there are no values filling the rest of the inputs, the inputs and the output will be left as uninstantiated.

Area/Resources Information

Elaborated Design



Synthesis / Implementation Resource Usage

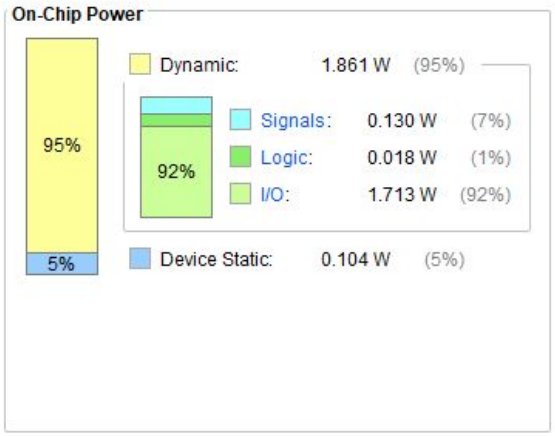
Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAMs	URAM	DSP
synth_1	constrs_1	synth_design Complete!								4	0	0.0	0	0
impl_1	constrs_1	write_bitstream Complete!	NA	NA	NA	NA	NA	1.965	0	4	0	0.0	0	0

Power Usage

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**1.965 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**34.0°C**
Thermal Margin:	51.0°C (11.1 W)
Effective ϑJA:	4.6°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity



Post-Implementation Resource Utilization