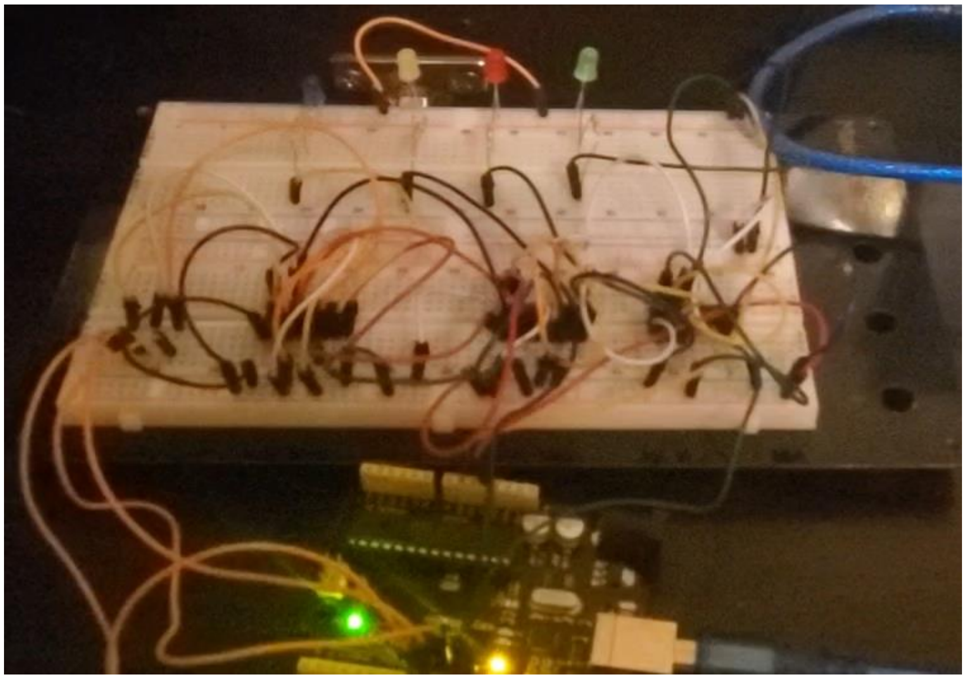
Logic Simplification & Implementation

ECE2300L Module 2 Report



Kyler Martinez

September 23rd, 2020

Introduction

The learning objectives of the module include learning how to simplify and implement logic to save costs or decrease the difficulty in implementation. To do this we explore reducing logic expressions with Boolean algebra further along with using Karnaugh maps to simplify complete and incompletely specified scenarios. Finally, we discuss contrasting various implementations of a combinational logic circuit using devices such as discrete gates, decoders, and multiplexors as well as how they compare to each other.

Activity 2.1 — Simplification using Boolean Algebra and Karnaugh Map

A. If F (A,B,C) = ABC’ + AB’ + A’BC + ABC + A’B’C’, then the minimum logic gates that would be used to implement the function, if we could have a maximum of 3 inputs, would be 10. The circuit would be composed of two 3-input OR gates, four three input AND gates, one two input AND gate, and three NOT gates.

B. Simplifying F with Boolean algebra

F (A,B,C) = ABC’ + AB’ + A’BC + ABC + A’B’C’
BC(A’ + A) + ABC’ + A’B’C’ + AB’
BC + ABC’ + A’B’C’ + AB’
B(C + AC’) + B’ (A + A’C’)
B(C+A)(C+C’) + B’(A+A’)(A+C’)
B(C+A) + B’(A+C’)
AB+AB’+BC+B’C’
A(B+B’) + BC + B’C’
<u>F(A,B,C) = A + BC + B’C’</u>

C. Using the simplified logic expression for F, we can find that the minimum logic gates need would be 5 in total. The circuit would be composed of one three input OR gate, two NOT gates, and two two-input AND gates.

D. To expand the original expression of F into a sum of minterms we notice that every term except AB’ has each variable included in the term. Thus, to incorporate C into the term we must expand the term without changing the expression. Due to the fact that X(1) = X and (X’+ X) =1 then we can reform AB’ to be AB’(C’ + C) with turns into AB’C + AB’C’. Thus the expanded canonical expression is: <u>F (A,B,C) = ABC’ + AB’C + AB’C’ + A’BC + ABC + A’B’C’</u>

E.

<u>Truth Table of F(A,B,C)</u>

ABC	Minterms	F(A,B,C)
000	A’B’C’	1
001	A’B’C	0
010	A’BC’	0
011	A’BC	1
100	AB’C’	1
101	AB’C	1
110	ABC’	1
111	ABC	1

F. K-Map For F= (A,B,C)

A\BC	00	01	11	10
0	1	0	1	0
1	1	1	1	1

<u>F(A,B,C) = A + BC + B’C’</u>

Activity 2.2 — Functions With Don’t Care Outputs

BCD Number System Table With BCD As Input And BCD + 1 As Output

Decimal	BCD ($B_3B_2B_1B_0$)	BCD+1 ($D_3D_2D_1D_0$)
0	0000	0001
1	0001	0010
2	0010	0011
3	0011	0100
4	0100	0101
5	0101	0110
6	0110	0111
7	0111	1000
8	1000	1001
9	1001	0000
10	1010	xxxx
11	1011	xxxx
12	1100	xxxx
13	1101	xxxx
14	1110	xxxx
15	1111	xxxx

A.

K-Map For D_3

B_3B_2\ B_1B_0	00	01	11	10
00	0	0	0	0
01	0	0	1	0
11	x	x	x	x
10	1	0	x	x

$D_3 = B_3B_1'B_0' + B_2B_1B_0$

K-Map For D_2

B_3B_2\ B_1B_0	00	01	11	10
00	0	0	1	0
01	1	1	0	1
11	x	x	x	x
10	0	0	x	x

$D_2 = B_2B_1' + B_3'B_2'B_1B_0 + B_2B_1B_0'$

K-Map For D_1

B_3B_2\ B_1B_0	00	01	11	10
00	0	1	0	1
01	0	1	0	1
11	x	x	x	x
10	0	0	x	x

$D_1 = B_3'B_1'B_0 + B_1B_0'$

K-Map For D_0

B_3B_2\ B_1B_0	00	01	11	10
00	1	0	0	1
01	1	0	0	1
11	x	x	x	x
10	1	0	x	x

$D_0 = B_0'$

B. $D_3 = B_3B_1'B_0' + B_2B_1B_0$

$D_2 = B_2B_1' + B_3'B_2'B_1B_0' + B_2B_1B_0'$

$D_1 = B_3'B_1'B_0 + B_1B_0'$

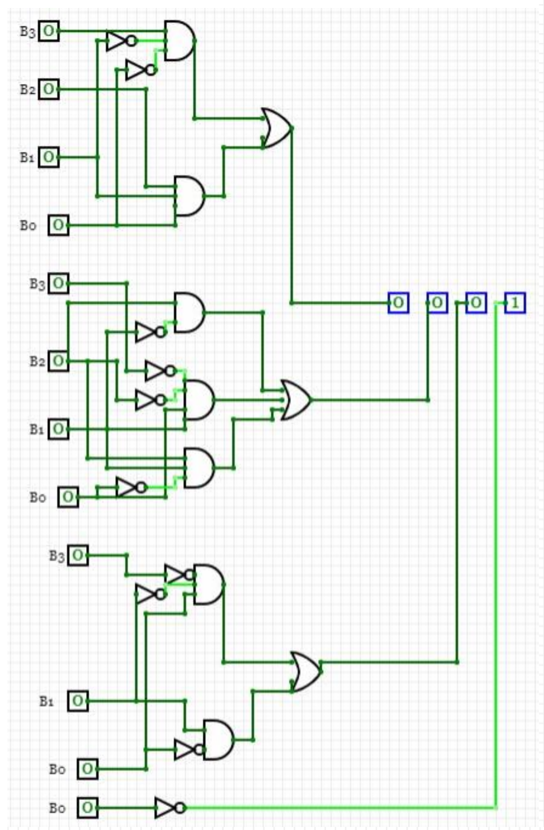
$D_0 = B_0'$

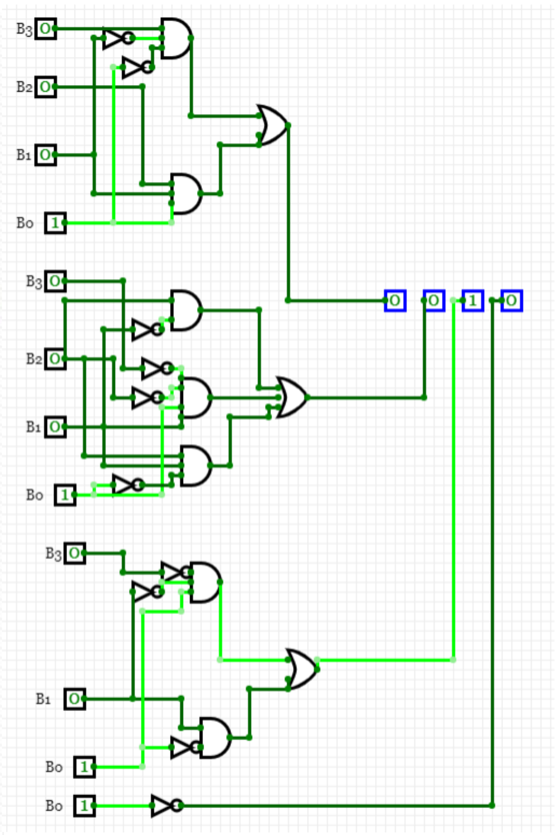
C.

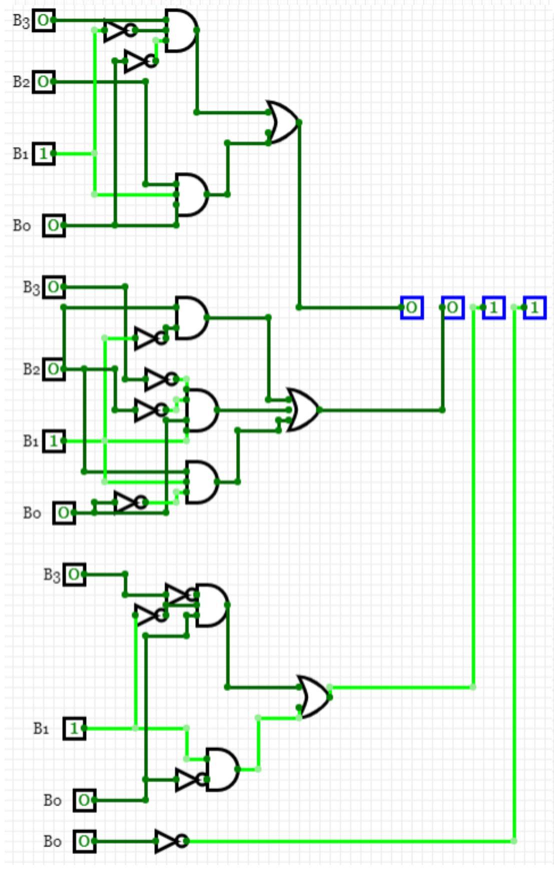
For input combinations greater than 9 there is a great likely hood that the output would be a repeated output for a previous combination however there is a chance that it is unique if the output is greater than binary 1001.The output that has the largest deciding factor is D_3 since if D_3 is not equal to 1, then the output combination will be a repeat, however if both D_3 is equal to one and D_2 and D_1 equal zero, then it will also be a repeat. This is not an issue since we do not concern ourselves with the outputs greater than decimal 9 since we can not use these in the decimal system.

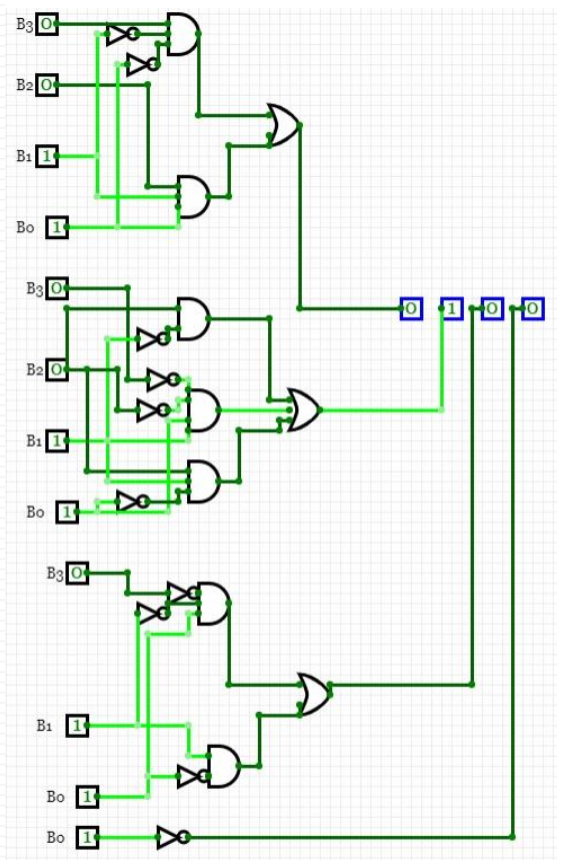
D.

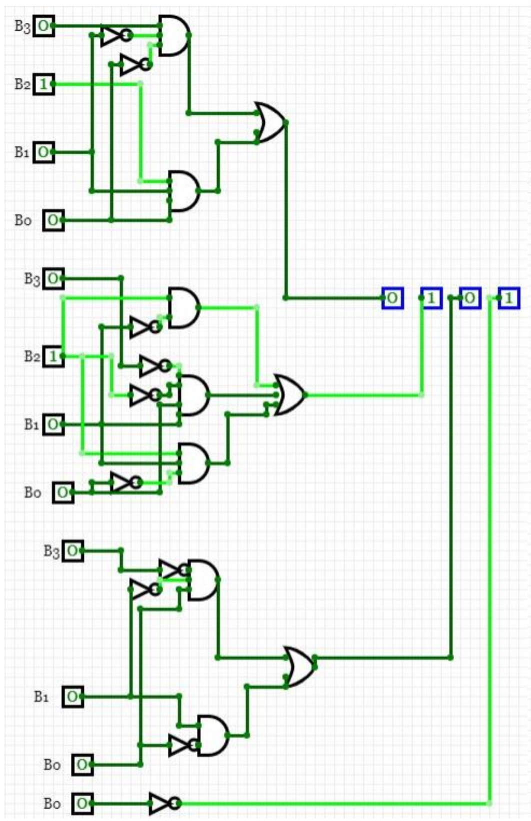
To verify our simplifications, we used CircuitVerse to implement the expressions and compare our outputs to the outputs in the provided table. For ease of visibility, each expression was given their own set of inputs, but were changed to match the variable they were representing.

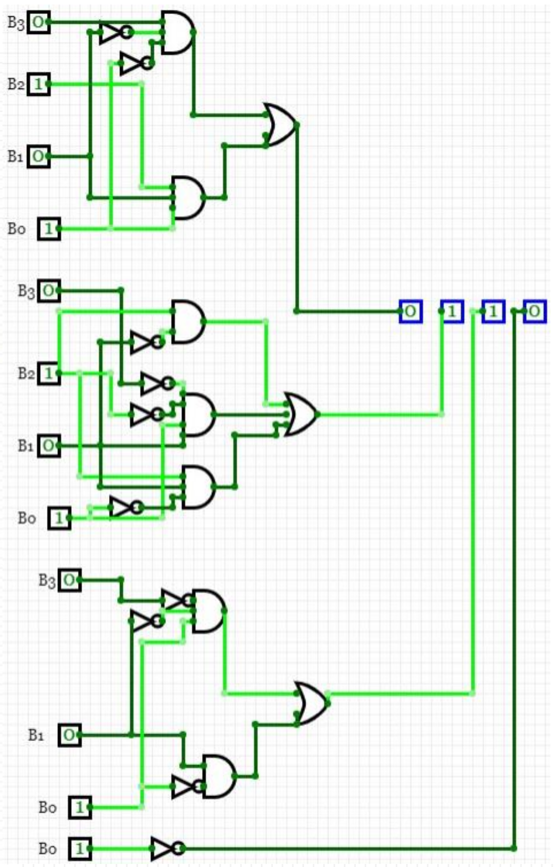

B3 0
B2 0
B1 0
Bo 0
B3 0
B2 0
B1 0
Bo 0
B3 0
B1 0
Bo 0
Bo 0
0 0 0 1

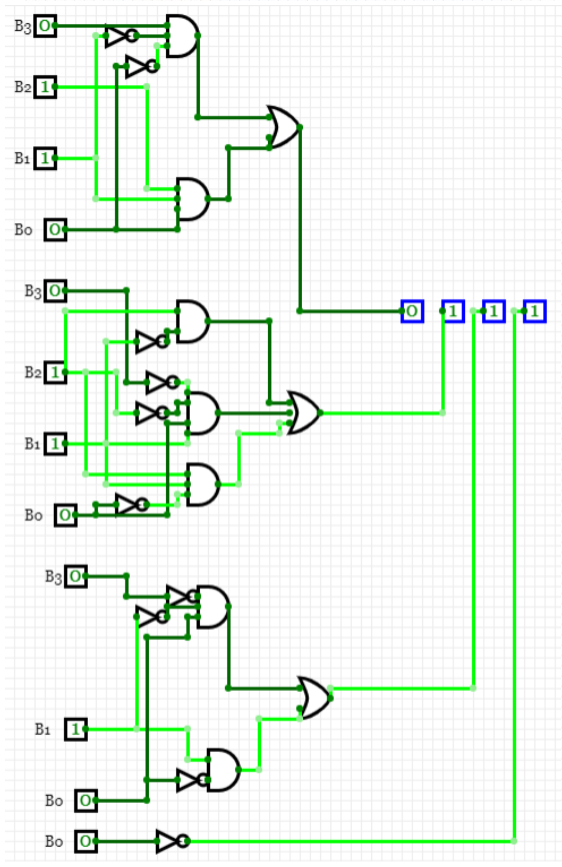

B3 0
B2 0
B1 0
Bo 1
B3 0
B2 0
B1 0
Bo 1
B3 0
B1 0
Bo 1
Bo 1
0 0 1 0

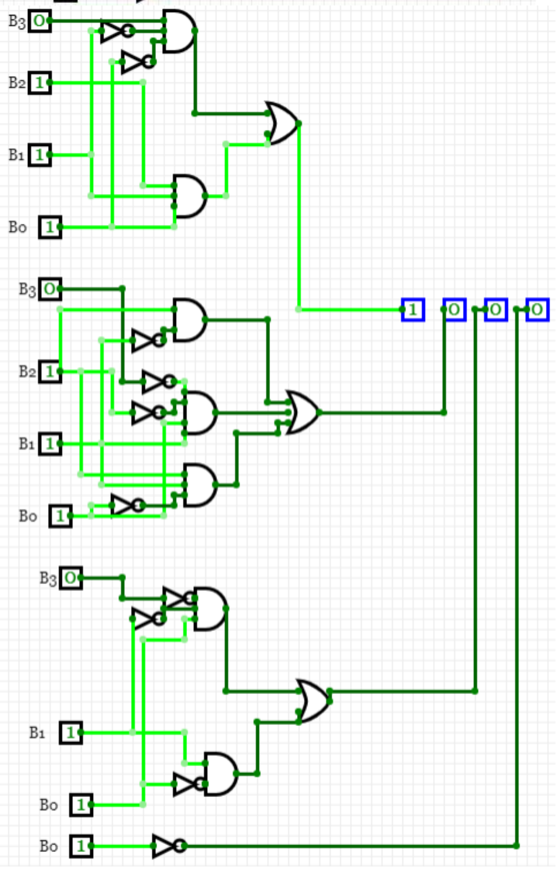

B3 0
B2 0
B1 1
Bo 0
B3 0
B2 0
B1 1
Bo 0
B3 0
B1 1
Bo 0
Bo 0
0 0 1 1

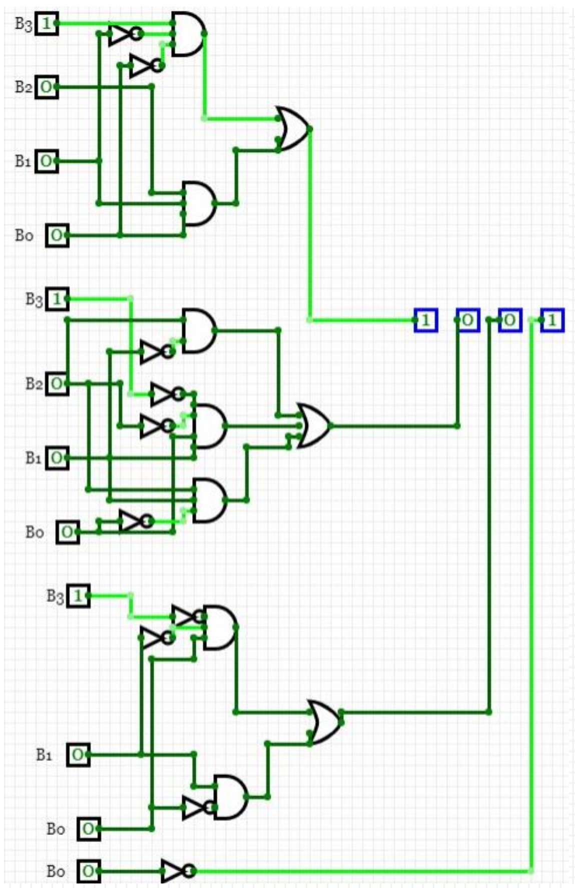

B3 0
B2 0
B1 1
Bo 1
B3 0
B2 0
B1 1
Bo 1
B3 0
B1 1
Bo 1
Bo 1
0 1 0 0

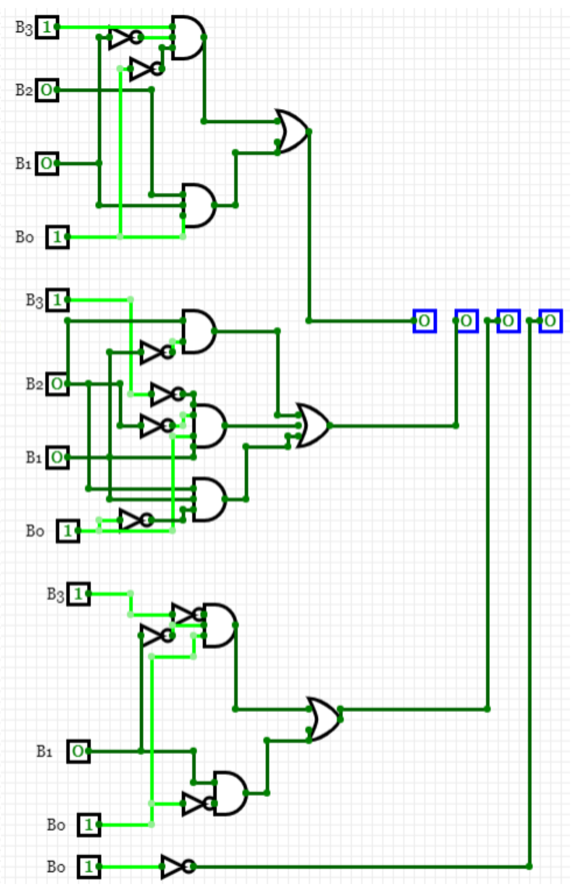

B3 0
B2 1
B1 0
Bo 0
B3 0
B2 1
B1 0
Bo 0
B3 0
B1 0
Bo 0
Bo 0
0 1 0 1

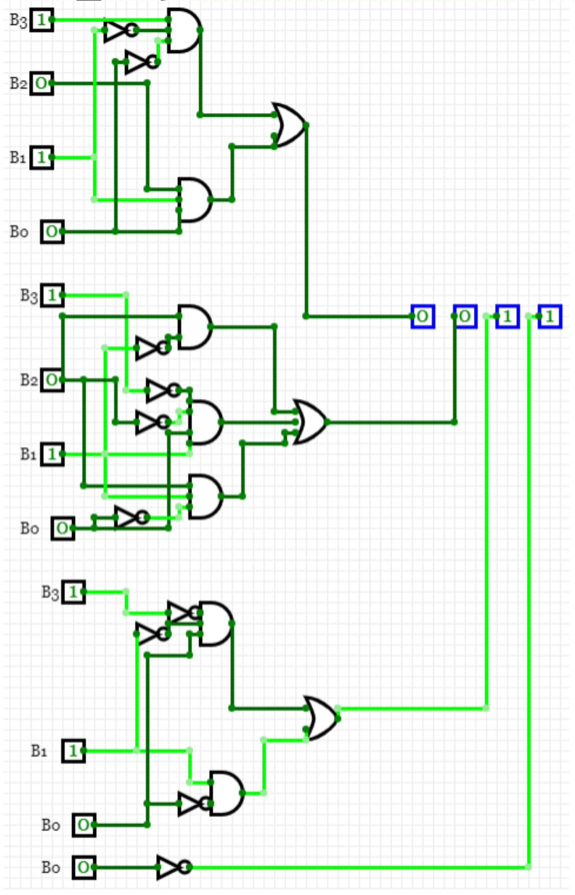

B3 0
B2 1
B1 0
Bo 1
B3 0
B2 1
B1 0
Bo 1
B3 0
B1 0
Bo 1
Bo 1
0 1 1 0

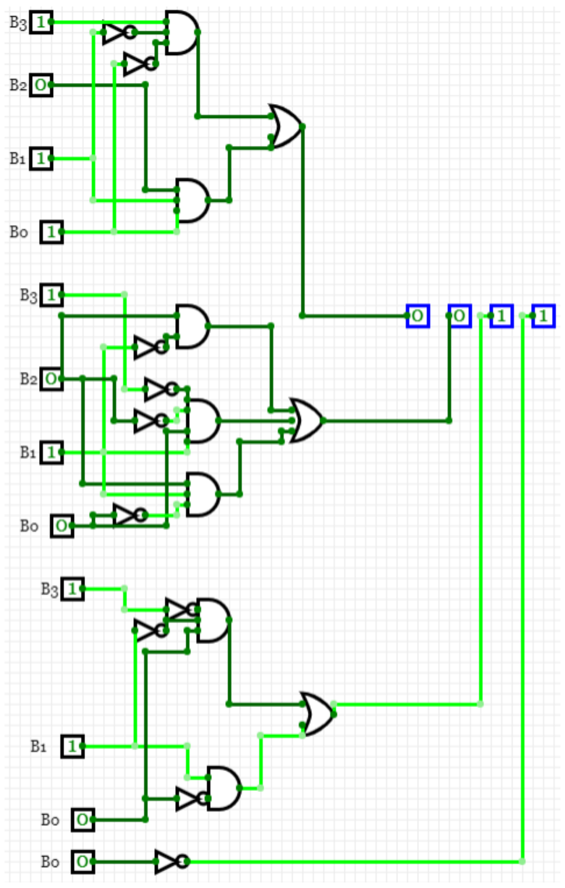

B3 0
B2 1
B1 1
Bo 0
B3 0
B2 1
B1 1
Bo 0
B3 0
B1 1
Bo 0
Bo 0
0 1 1 1

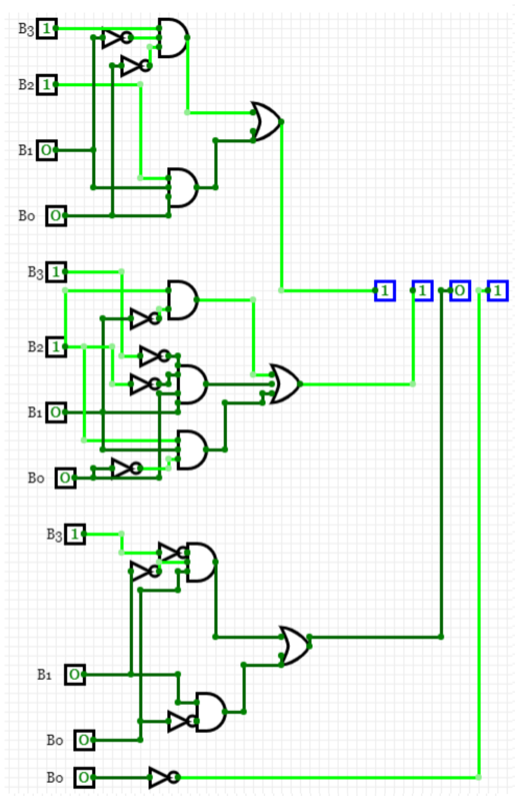

B3 0
B2 1
B1 1
Bo 1
B3 0
B2 1
B1 1
Bo 1
B3 0
B1 1
Bo 1
Bo 1
1 0 0 0


B3 1
B2 0
B1 0
Bo 0
B3 1
B2 0
B1 0
Bo 0
B3 1
B1 0
Bo 0
Bo 0
1 0 0 1

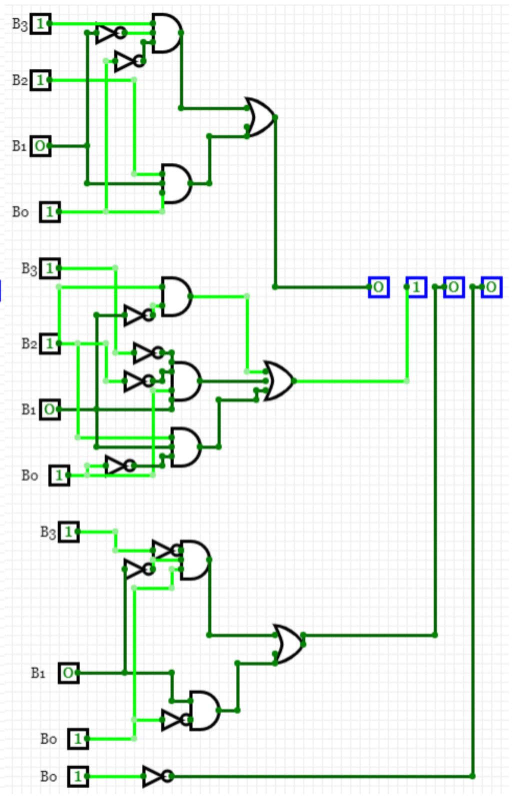

B3 1
B2 0
B1 0
Bo 1
B3 1
B2 0
B1 0
Bo 1
B3 1
B1 0
Bo 1
Bo 1
0 0 0 0


B3 1
B2 0
B1 1
Bo 0
B3 1
B2 0
B1 1
Bo 0
B3 1
B1 1
Bo 0
Bo 0
0 0 1 1

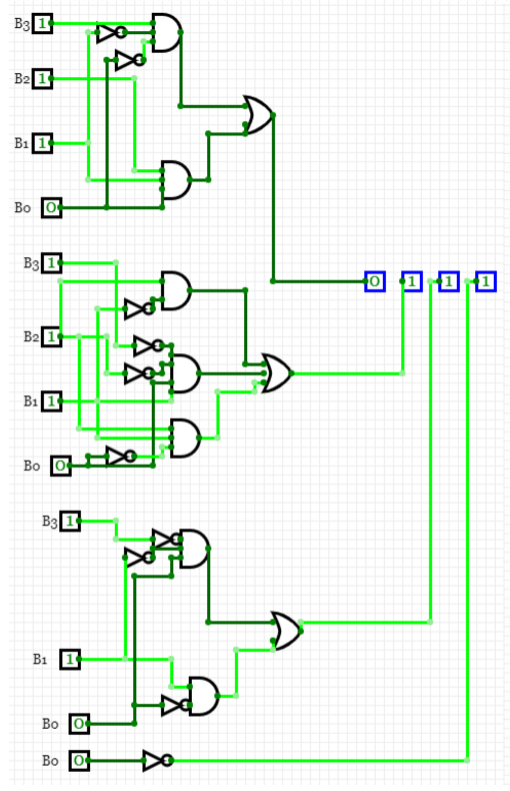

B3 1
B2 0
B1 1
Bo 1
B3 1
B2 0
B1 1
Bo 0
B3 1
B1 1
Bo 0
Bo 0
0 0 1 1



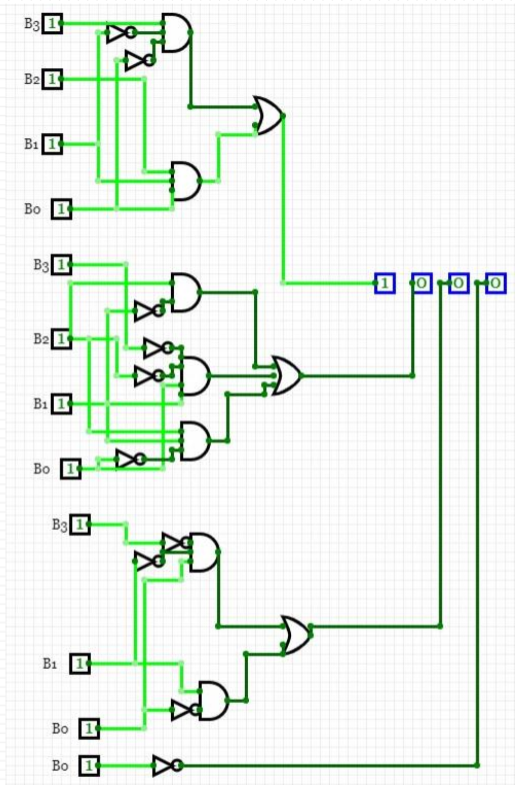
B3 1
B2 1
B1 0
Bo 0
B3 1
B2 1
B1 0
Bo 0
B3 1
B1 0
Bo 0
Bo 0
1 1 0 1



B3 1
B2 1
B1 0
Bo 1
B3 1
B2 1
B1 0
Bo 1
B3 1
B1 0
Bo 1
Bo 1
0 1 0 0



B3 1
B2 1
B1 1
Bo 0
B3 1
B2 1
B1 1
Bo 0
B3 1
B1 1
Bo 0
Bo 0
0 1 1 1



B3 1
B2 1
B1 1
Bo 1
B3 1
B2 1
B1 1
Bo 1
B3 1
B1 1
Bo 1
Bo 1
1 0 0 0

Activity 2.3 — Functions With Don't Care Inputs

The programming statement if (S==1) M= B; else M = A; can be interpreted as a Boolean expression if S,M,B, and A are single-bit Boolean variables

A.

M(S,A,B) Truth Table

S	A	B	M
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

B.

To describe the truth table in a more concise format, you need to be able to find variables that no longer impact the output when certain conditions are met. When S is equal to zero, then only A influences the output so B becomes a don't care input and we can condense four rows into two, one for each possibility of A. When S is equal to one, then only B affects the output of M, thus A becomes a don't care input, and we can then condense four rows into two.

M(S,A,B) Condensed Truth Table

S	A	B	M
0	0	x	0
0	1	x	1
1	x	0	0
1	x	1	1

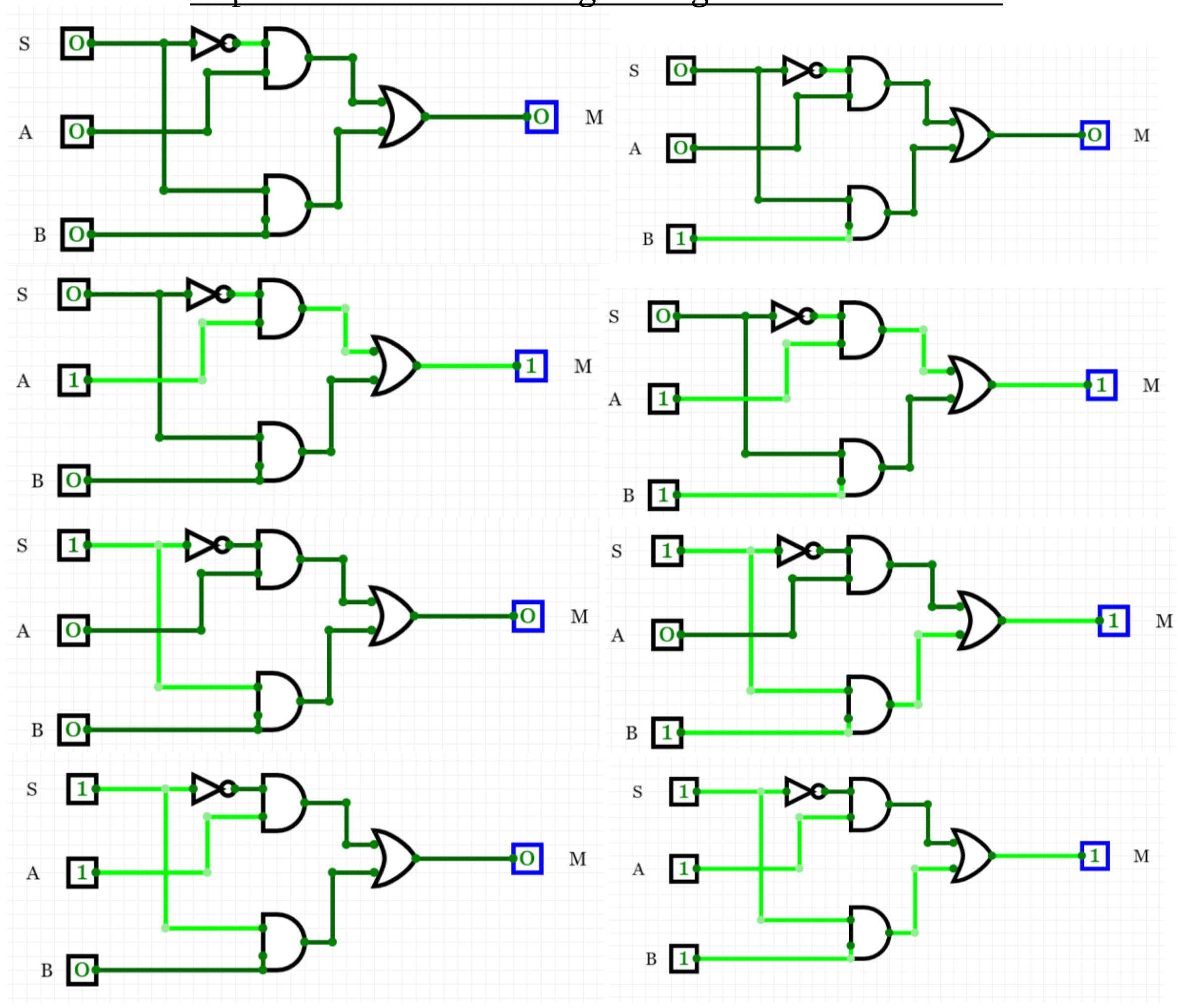
C.

To derive the Boolean expression, you can focus on the rows that follow the two conditions we defined. We can then create a sum of products expression following the rows where M is equal to one. Since we are using a function table, the inputs that are don't care are omitted from the expression since their values will not affect the output. These variables will be eliminated through simplification if we choose to include them.

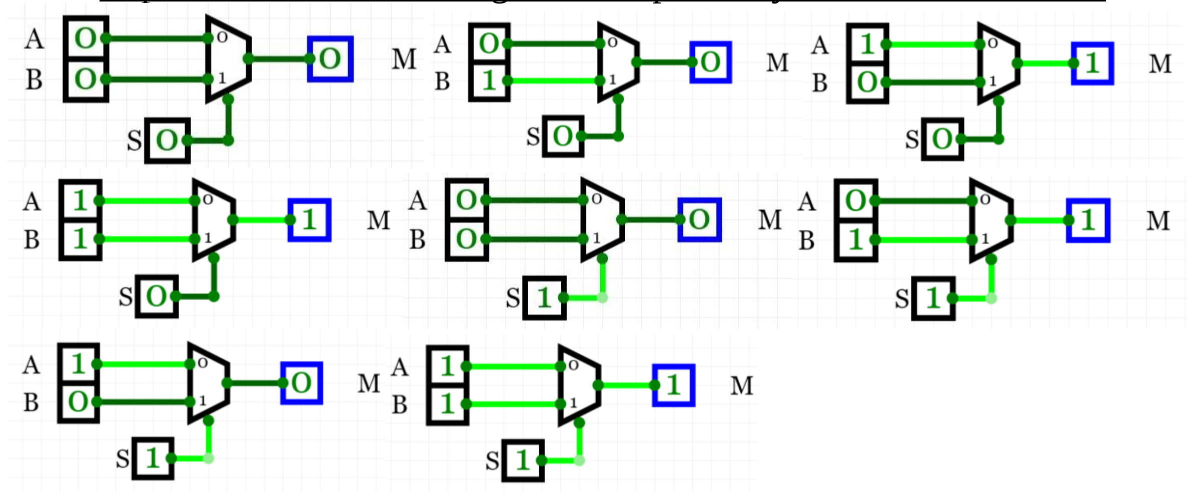
$$M(S,A,B) = SB + S'A$$

D. .

<u>Implementation of M using basic gates in CircuitVerse</u>



<u>Implementation of M using the multiplexor symbol in CircuitVerse</u>



Activity 2.4 — Logic Implementation With Multiplexers

Problem Statement: Design a machine that outputs true if a valid combination of coffee, tea, and milk is inputted. These valid choices are coffee alone, tea alone, milk alone, coffee and milk, and tea and milk.

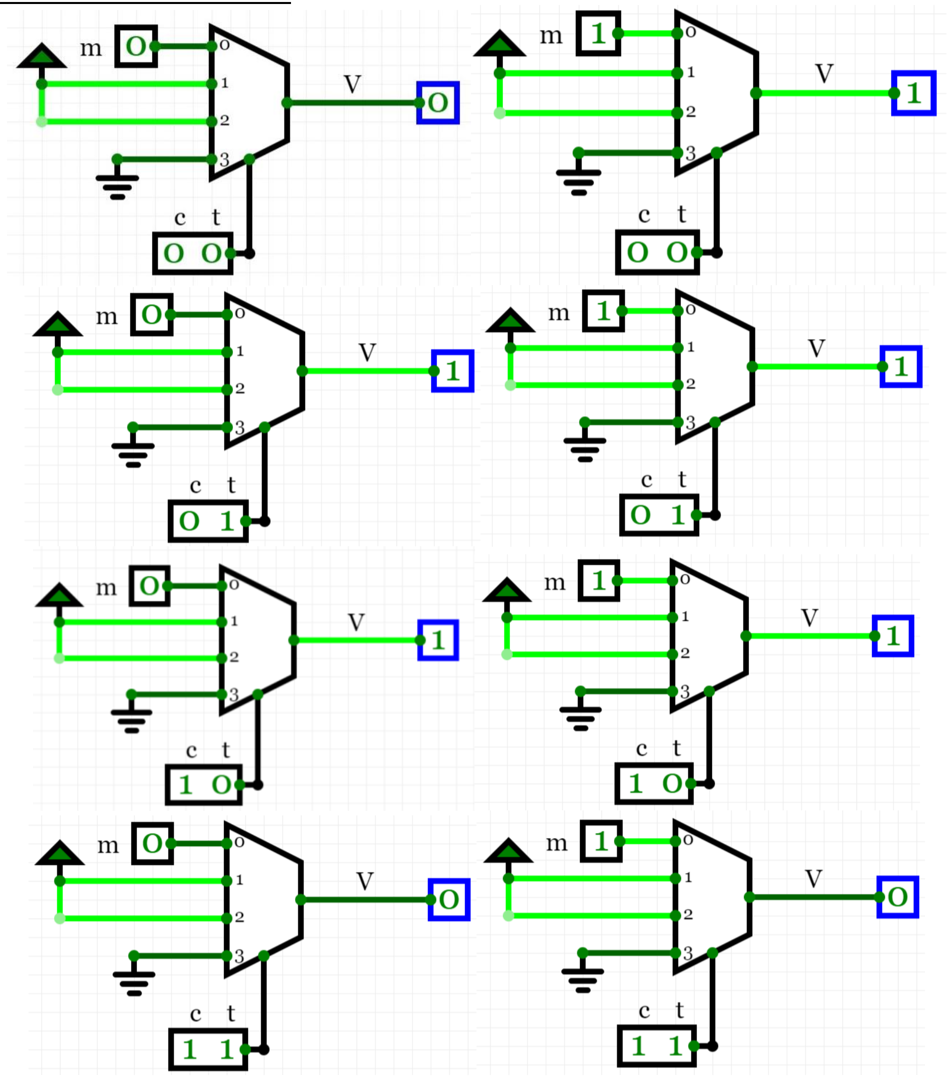
A.

Truth Table For The Machine

c	t	m	V
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

B.

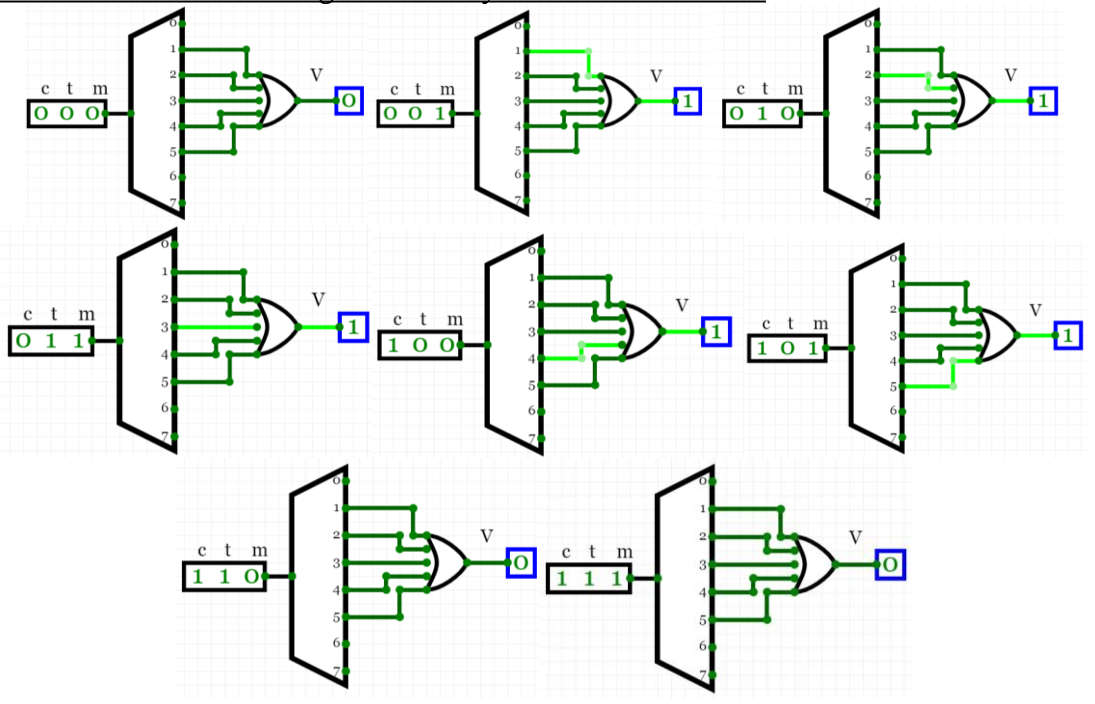
Implementation In CircuitVerse



Activity 2.5 — Logic Implementation With Decoders

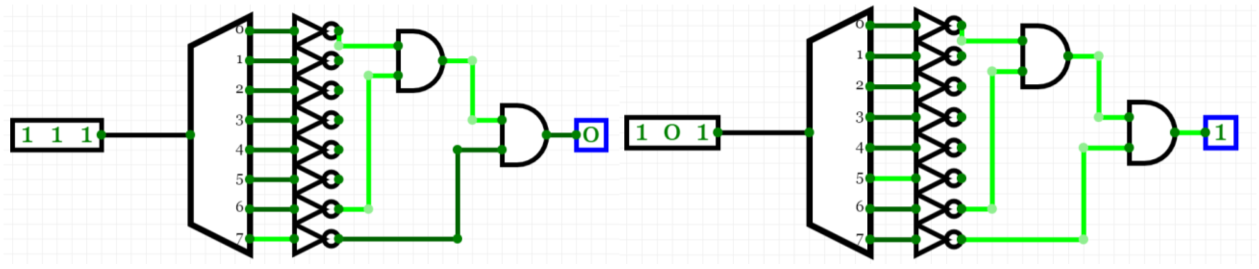
A.

Implementation Of The Design In Activity 2.4 With A Decoder



B.

To implement the circuit design using the 74LS138, posed an interesting problem since the selected output would be at logic low rather than high, along with the restriction of using only one other device. A multi-input NAND gate could be used; however, I do not have access to one of those so there would not be one device that would be able to invert the outputs and then OR them together. The approach I took then was to do the opposite of what we would typically do. Since the output is essentially the inverse of what we would typically output. Since the outputs are logic high except for the selected output, AND gates can be used to connect all the outputs that would not be used if we had used an OR gate. This results in an output of logic level high when the output is not one of those connected to the AND gates, our desired outputs, and logic level low if one of the undesired outputs is selected since it would result in those outputs being considered low and in turn resulting in an overall logic level low for the output.



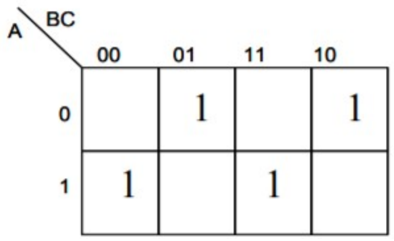
Video Link For 2.4 and 2.5: https://youtu.be/8xm4jUxg5bc

Note: I added the inverters so the decoder would output low, and this allow for me to show that the implemntation would work for the 74LS138. The mp4 file was also uploaded incase that was the preferred method.

Exercise 2.1 — Logic Simplification Revisited

1. The property of Boolean Algebra that the K-maps operate on is the various redundant variable theorems since the layout of the map have minterms that differ by one bit next to each other. Thus when there are two true minterms next to each other there will be a bit that is different, which means that there is a variable in the minterms that has the compliment in the other, which can then be reduced and removed from the expression.

2. For a K-map with a checkboard layout of 1's, seen in the right diagram, we cannot use the typical simplification that the K-map provides. However, you notice that there is a one when only one input is logic one, when all are logic one, and when A and B are logic one while C is logic zero. This would lead you to assume that the Boolean operation used is XOR. f(1,1,1) = 1 does cause some error in how we perceive an XOR gate, however you must see that the three input XOR gate can be made of two two-input XOR. Due to this we can see that as A and B would first go through an XOR gate and then through a final XOR gate with C, this allows for f(1,1,1)= 1, since the first XOR gate will return a logic 0, however the final gate will return logic 1 since C is at logic one.



3.

Truth Table For Logic Circuit

Decimal	BCD ($B_3B_2B_1B_0$)	bt9	Decimal	BCD ($B_3B_2B_1B_0$)	bt9
0	0000	0	8	1000	0
1	0001	0	9	1001	0
2	0010	0	10	1010	1
3	0011	0	11	1011	1
4	0100	0	12	1100	1
5	0101	0	13	1101	1
6	0110	0	14	1110	1
7	0111	0	15	1111	1

Functional Table

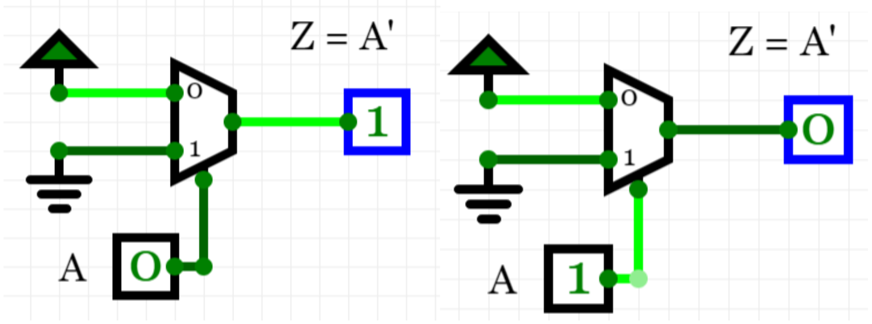
BCD ($B_3B_2B_1B_0$)	bt9
0xxx	0
100x	0
101x	1
11xx	1

A functional table works to help find a simplified expression without a K-map since a functional table works similarly to a K-map. When converting a functional table into an expression, the only variables included in a term are the ones that are not don't care inputs. When creating a functional table, the more rows that are combined, the less literals needed for the term that is included in the Boolean expression. This works similarly to K-maps grouping large amounts of 1's together and representing the group with a term with the product of the inputs that do not change over the region.

Exercise 2.2 — Multiplexor As Universal Gate

A.

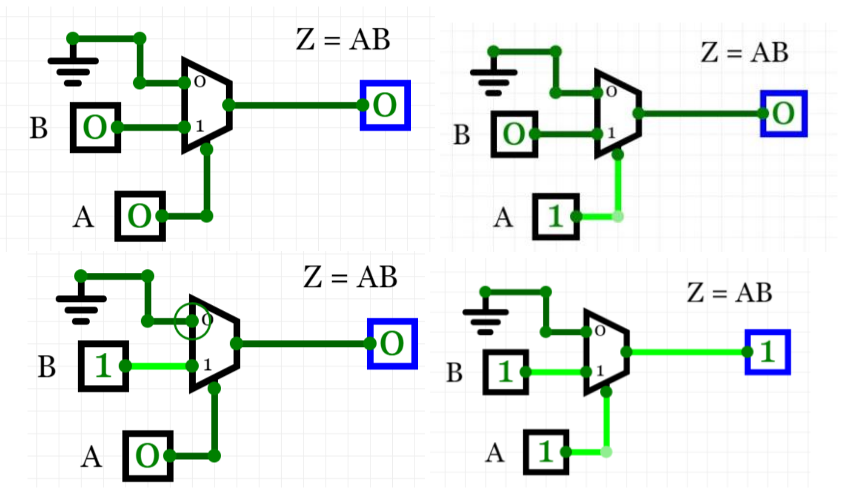
A 2-to-1 multiplexor can be modeled as a NOT gate as seen below:



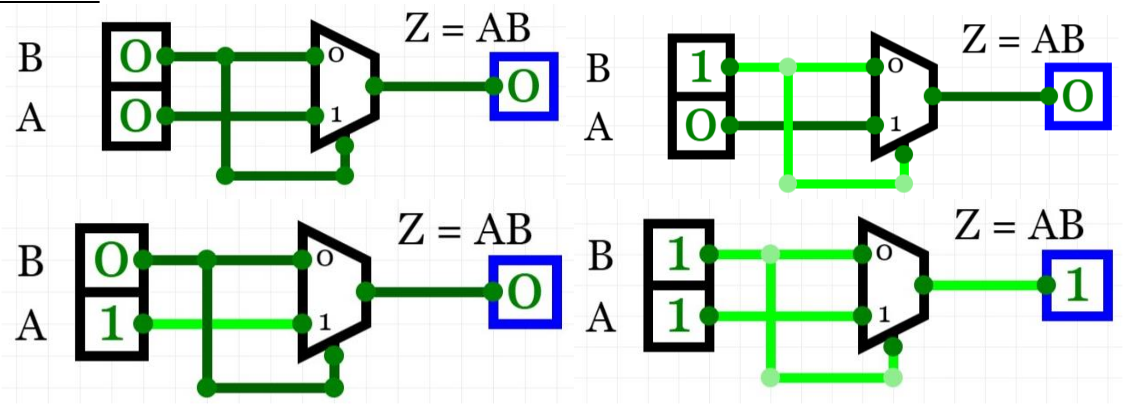
B.

A 2-to-1 multiplexor can be modeled as an AND gate in two ways:

Method 1:



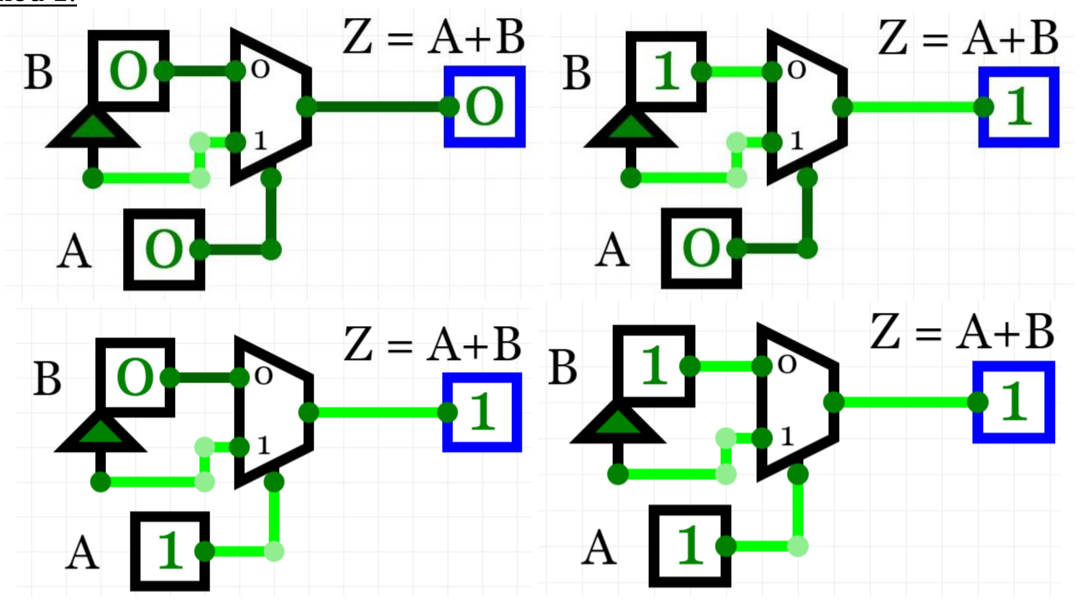
Method 2:



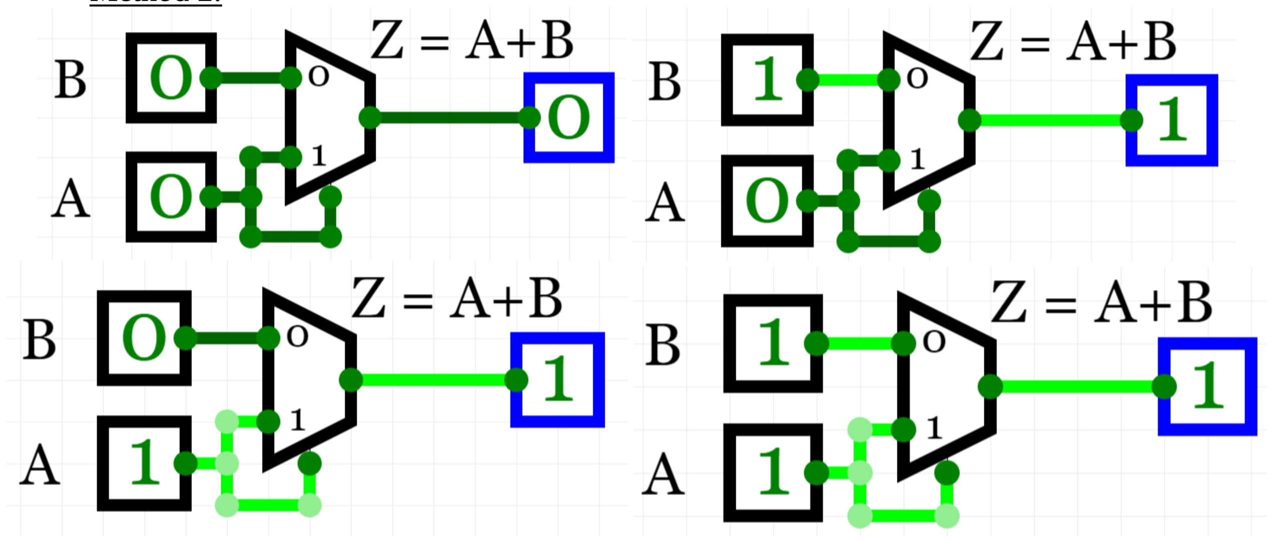
C.

A 2-to-1 multiplexor can be modeled as a OR gate in two ways:

Method 1:



Method 2:

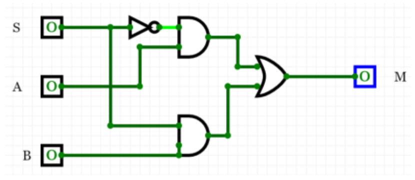


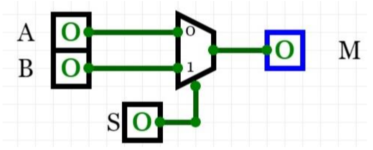
Exercise 2.3 - Comparison Among The Implementation Choices

The biggest factor that would determine what I would use would probably be whether I am analyzing the design on theoretical level, or if I am implementing the circuit using physical devices. I would use basic gates if the simplification allowed for easy analysis of the design. However I would not implement a circuit using basic gates since most circuits using basic gates would require multiple devices which would decrease the cost effectiveness and the time to build since you need to power three chips and then have a large enough breadboard to accompany those chips. This also allows for a greater chance of leaving gates unused. However, if a circuit would need an equal number of devices to be implemented with basic gates compared to other implementations, I would choose the basic gates due to general familiarity and simplicity.

I would use NAND/NOR gate implementation in a scenario where I can use as little chips as possible. In some implementations it is easier to use NAND gate implementation compared to basic gates since some NAND gates may become redundant in implementation and result in a fraction of chips necessary to implement the same circuit when compared to basic gates. NAND/NOR gate implementation would be avoided in circuits that are simple in design and may get more complicated or require more gates if implemented with NAND gates.

I would use a multiplexor in situations as shown in lab where I have inputs that have a sort of dominance in the selection of the output and result in inputs becoming don't care inputs. I would avoid using multiplexors as universal gates since they do not appear to simplify as cleanly as NAND gates tend to. However, I would use multiplexors to simplify a design using basic gates since the property for the multiplexor to simplify circuits allow for circuits needing multiple chips, containing all types of basic gates, to be condensed into one device. The circuits on the right are from Activity 2.3 and help showcase this point.





I would probably avoid decoders in implementing circuit designs unless I was going to use more of the outputs for the circuit if the implementation made special use of the min or max terms. Since a decoder would also need an OR or AND gate, I would choose to a different implementation if it could be done using less devices. Also, if the decoder is active low or active high, that could introduce other issues or be beneficial, however that would need to be determined on a case by case basis in order to see if a situation can be exploited.

Overall, I would stick with basic gates when analyzing the Boolean expression, however this is only because that is the method I am used to. I imagine I will cover a different method of analyzing and representing these circuits that is easier so I must comment on my current knowledge. For physical implementations I would use the method that requires the least work to configure since it would limit the number of devices used and decrease the likelihood of making an error when wiring multiple devices. This is mostly and issue when there are many wires on the board, and it becomes difficult to keep track of the wires and increase the likelihood of an error.