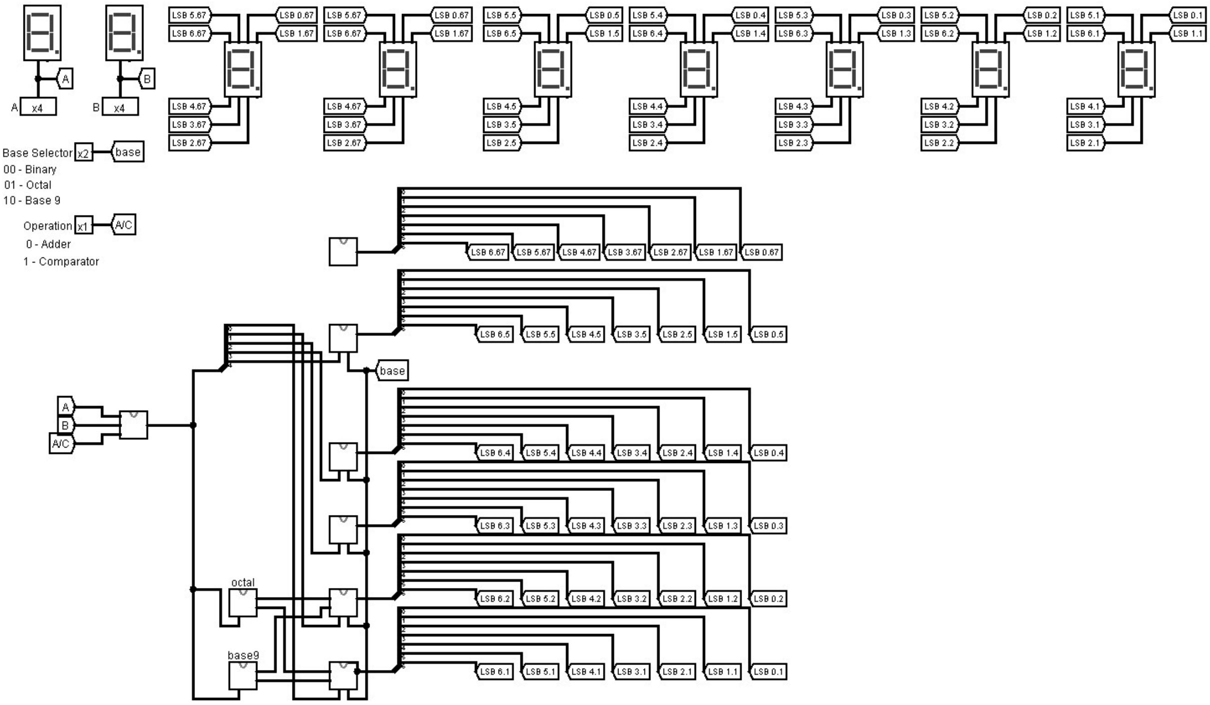
Documentation for CS 20 Project 1: Adder and Comparator with base conversion

Members:

Nicolas, Jack Vincent - 2020-03124 - M10-2

Ragunton, Carl David - 2020-04242 - M10-2

Circuit:



A x4
B x4
Base Selector x2 base
00 - Binary
01 - Octal
10 - Base 9
Operation x1 A/C
0 - Adder
1 - Comparator
A
B
A/C
base
octal
base9
LSB 6.67 LSB 5.67 LSB 4.67 LSB 3.67 LSB 2.67 LSB 1.67 LSB 0.67
LSB 6.5 LSB 5.5 LSB 4.5 LSB 3.5 LSB 2.5 LSB 1.5 LSB 0.5
LSB 6.4 LSB 5.4 LSB 4.4 LSB 3.4 LSB 2.4 LSB 1.4 LSB 0.4
LSB 6.3 LSB 5.3 LSB 4.3 LSB 3.3 LSB 2.3 LSB 1.3 LSB 0.3
LSB 6.2 LSB 5.2 LSB 4.2 LSB 3.2 LSB 2.2 LSB 1.2 LSB 0.2
LSB 6.1 LSB 5.1 LSB 4.1 LSB 3.1 LSB 2.1 LSB 1.1 LSB 0.1

Division of Tasks:

We did this project with no clear division of the tasks. We worked separately on the parts we know to do and told each other what parts of circuit was already done. Jack Nicolas worked primarily on the beginning of the circuit while Carl Ragunton made parts from the circuit's middle towards the end.

In the end of creating the circuit, here are the exact contributions made by the members:

Jack Nicolas:

- Adder

- Comparator

- Combined the parts and simplified it

- Sub circuit for the operation selector

Carl Ragunton:

- Base conversion to octal
- Base conversion to base-9/mystery base
- Sub circuits regarding the base selector and outputs

Short Narrative:

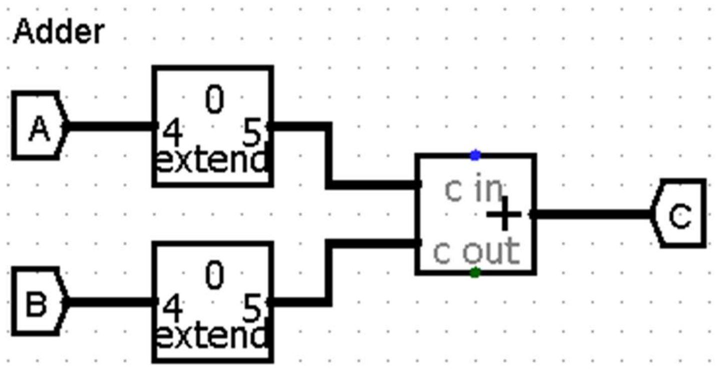
As stated on the division of tasks, we did not divide exactly what part each of us will make. Jack Nicolas started on creating the adder and comparator parts of the circuit. Carl Ragunton, then, made the base conversion to octal. We made use of the fact that 5 bits is the maximum number of bits that will be processed in the base conversion. The process was followed by the creation of the base converter to base-9/mystery base which was quite challenging at first because the use of divider was not allowed. Sub circuits involving the base selector and output was, then, made. Last part of the creation of this circuit is combining the parts and simplifying the wirings to make it look clean. Of course, we asked each other question regarding the parts we did not understand.

Details of Each Parts/Processes:

1. Adder

The main part of this function is the built-in adder function of Logisim. Since the two inputs are 4-bits the maximum number of bits that it can produce when added is 5-bits. By utilizing this information, we made use of bit extenders to turn the 4-bit input to 5-bit input and then connected into a 5-bit adder.

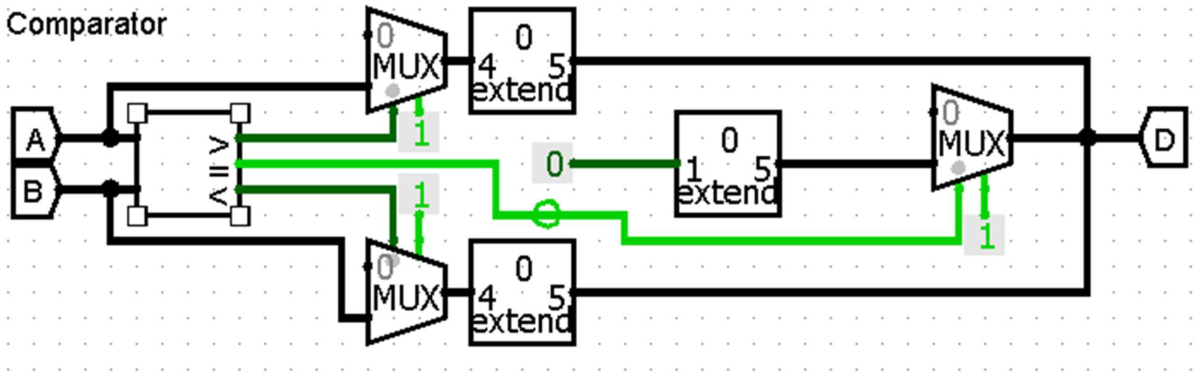
Here is the circuit used for the adder function:



2. Comparator

The main part of this function is the built-in comparator and multiplexer of Logisim. Since the built-in comparator only outputs 1-bit whether it is <,>, or =, we used a multiplexer to produce the desired output. By making use of the 1-bit output of the comparator, it connects to the multiplexer which will check if it will output the first input, the second output or simply 0. We also made use of a bit extender to change the 4-bit output into a 5-bit output so that it can be utilized later on.

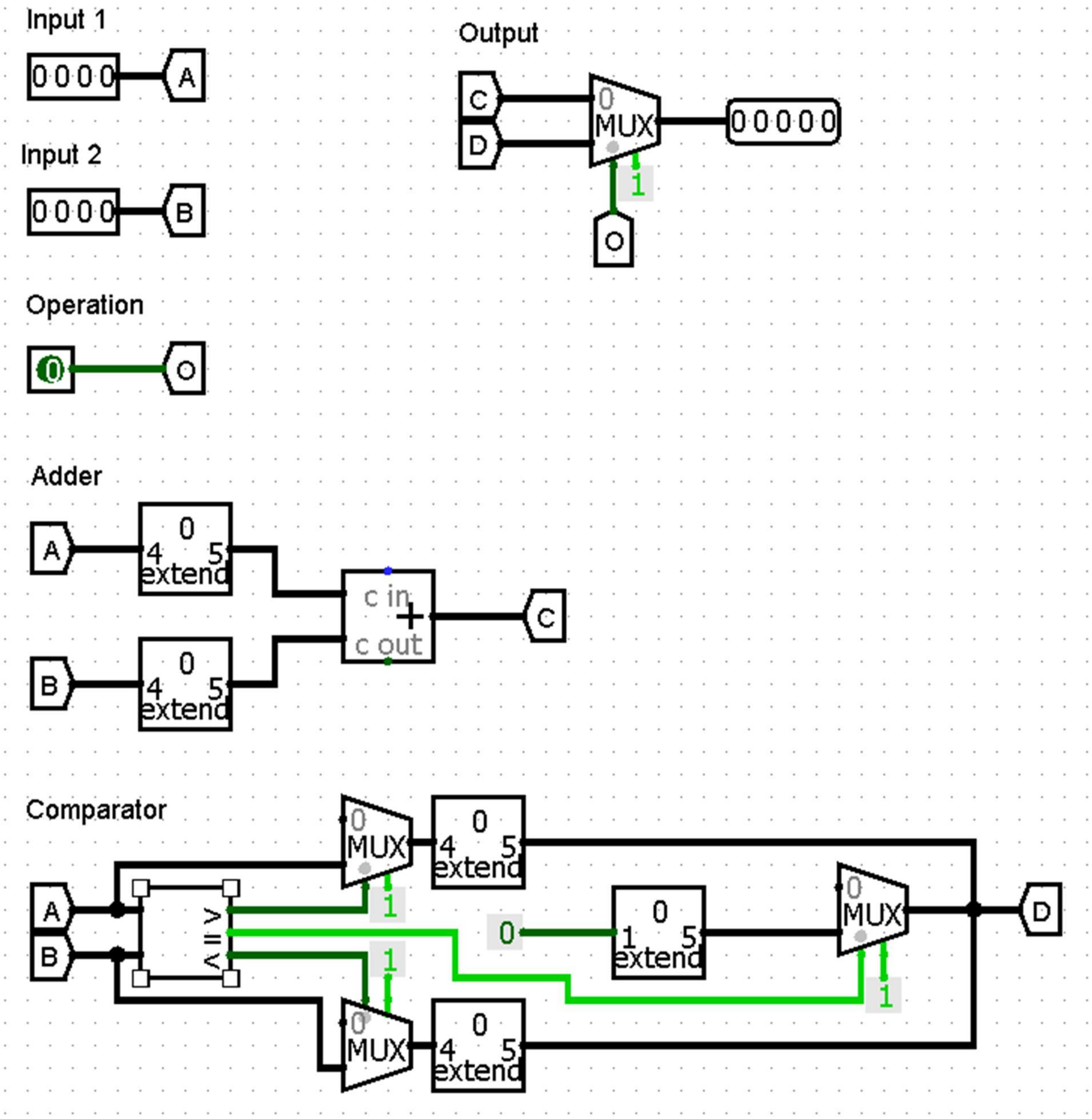
Here is the circuit used for the comparator function:



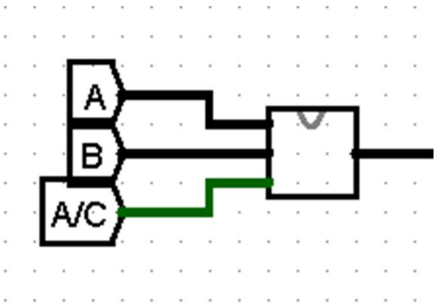
3. Sub circuit regarding the operation selector

In this sub circuit, the adder and comparator function is joined together but only functions depending on the input on which operator to use. Other than the initial two 4-bit inputs, there is an additional 1-bit input which denotes the operation to use (0 – Adder, 1 – Comparator). The output uses a multiplexer to denote which will be outputted to the main circuit.

Here is the whole sub circuit of the operation selector:



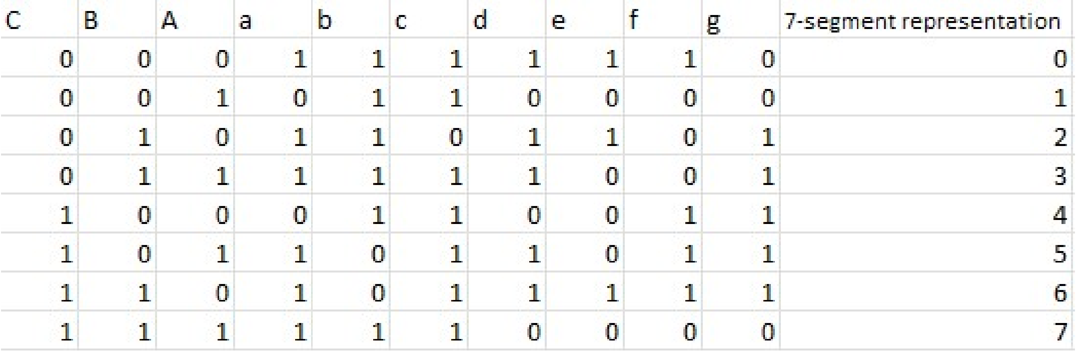
This is how it is shown in the main circuit:



4. Base Conversion to Octal

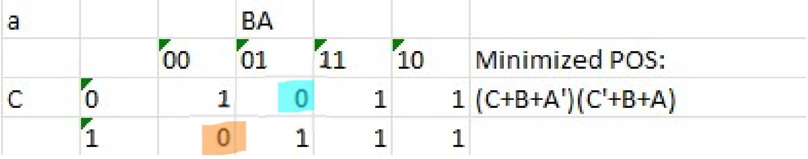
The key part in this part is that the maximum number of bits that needed to be processed is 5 bits. Note that binary can be converted into octal by grouping them into 3 bits. That means that the last three bits will be the LSB octal and the first 2 bits will be part of the 2nd LSB octal. We just added a constant 0 to be part of the latter. For now, we just need to worry about the 2 LSB outputs because the rest will just become zeroes.

This part was already done before it was announced that Logisim's combinational analysis function can be used. Therefore, the first step is to make the truth table which is here:

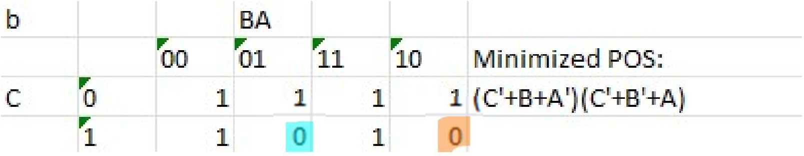


C	B	A	a	b	c	d	e	f	g	7-segment representation
0	0	0	1	1	1	1	1	1	0	0
0	0	1	0	1	1	0	0	0	0	1
0	1	0	1	1	0	1	1	0	1	2
0	1	1	1	1	1	1	0	0	1	3
1	0	0	0	1	1	0	0	1	1	4
1	0	1	1	0	1	1	0	1	1	5
1	1	0	1	0	1	1	1	1	1	6
1	1	1	1	1	1	0	0	0	0	7

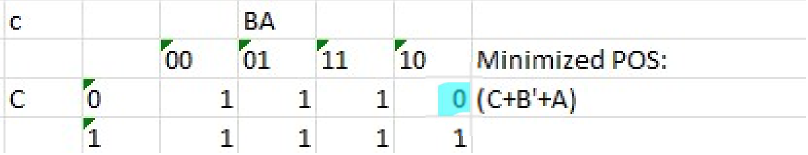
After that, k-maps were used to find the minimized POS of each output of the 7-segment decoder. Here are the k-maps and the minimized POS:



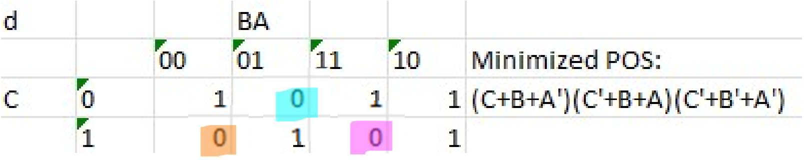
a			BA			
		00	01	11	10	Minimized POS:
C	0	1	0	1	1	(C+B+A')(C'+B+A)
	1	0	1	1	1	



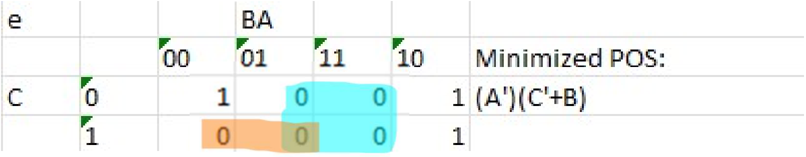
b			BA			
		00	01	11	10	Minimized POS:
C	0	1	1	1	1	(C'+B+A')(C'+B'+A)
	1	1	0	1	0	



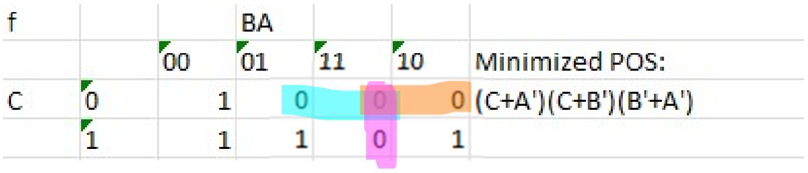
c			BA			
		00	01	11	10	Minimized POS:
C	0	1	1	1	0	(C+B'+A)
	1	1	1	1	1	



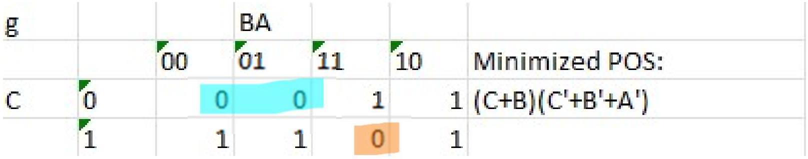
d			BA			
		00	01	11	10	Minimized POS:
C	0	1	0	1	1	(C+B+A')(C'+B+A)(C'+B'+A')
	1	0	1	0	1	



e			BA			
		00	01	11	10	Minimized POS:
C	0	1	0	0	1	(A')(C'+B)
	1	0	0	0	1	

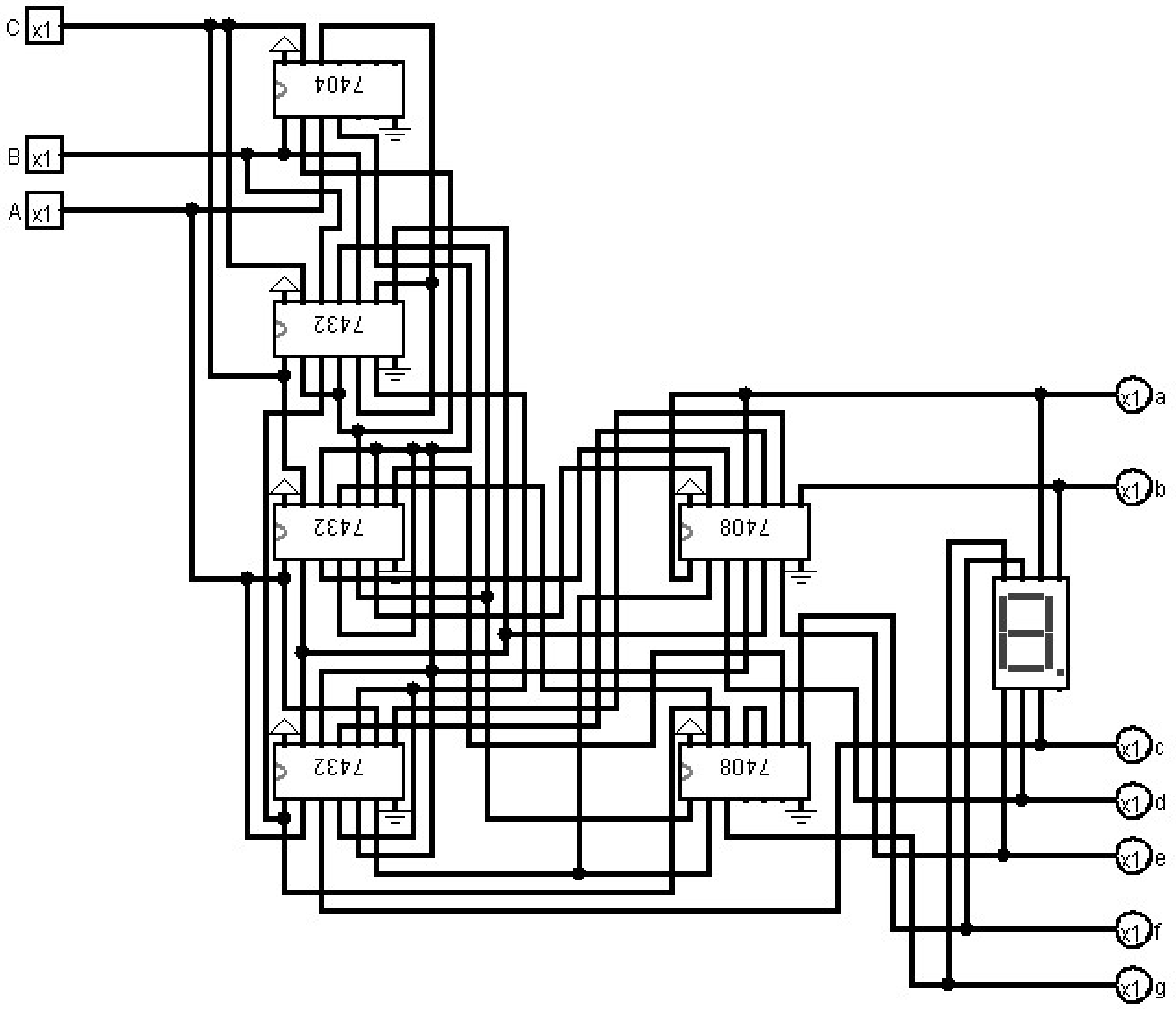


f			BA			
		00	01	11	10	Minimized POS:
C	0	1	0	0	0	(C+A')(C+B')(B'+A')
	1	1	1	0	1	

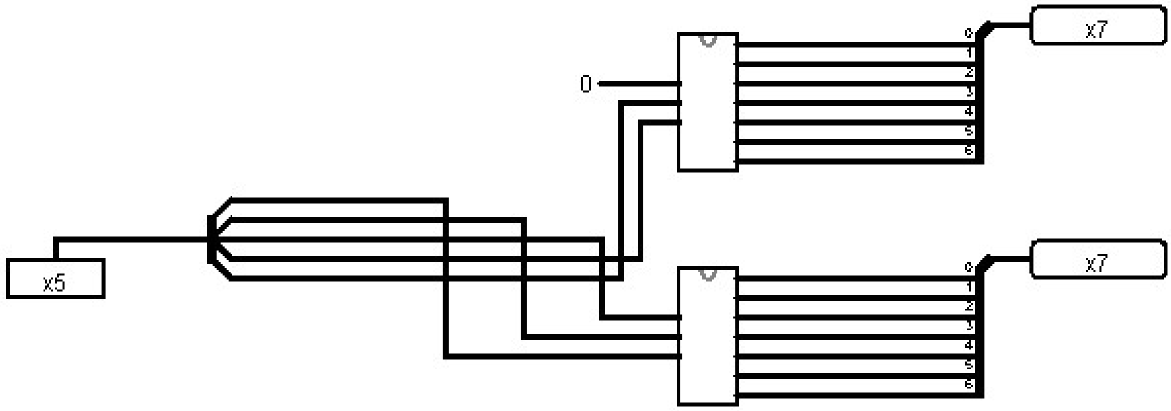


g			BA			
		00	01	11	10	Minimized POS:
C	0	0	0	1	1	(C+B)(C'+B'+A')
	1	1	1	0	1	

Now that we have a Boolean expression for each 7-segment decoder output, we can create the sub circuits. We used 7404 IC/NOT gates, 7432 IC/OR gates, and 7408 IC/AND gates to create the sub circuit below:



We created another sub circuit that will output the 2 LSB octal. This is just to make the final output look simpler.

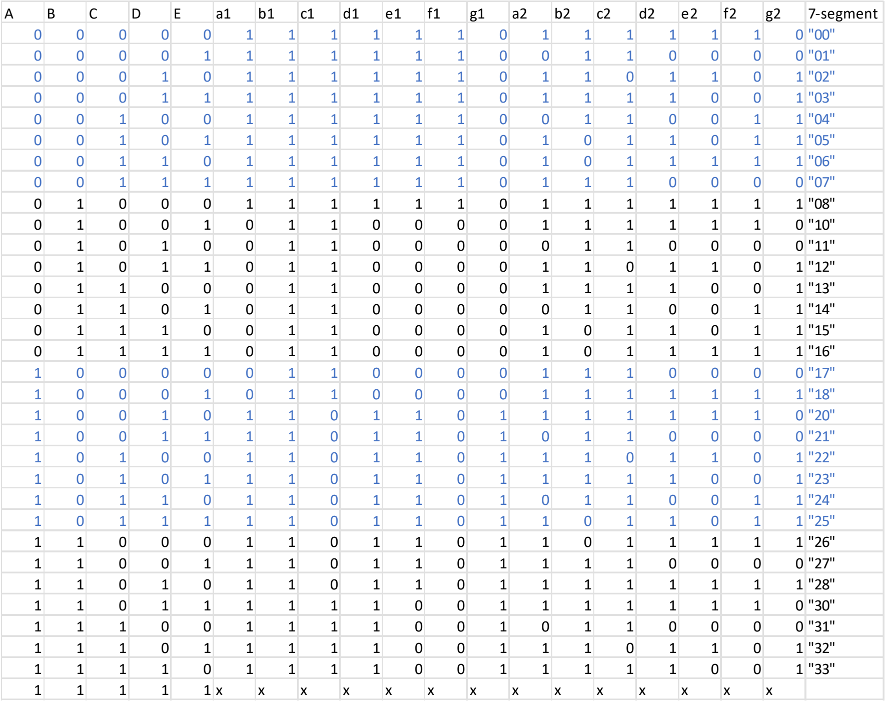


5. Base Conversion to Base-9/Mystery Base

Our mystery base is base-9. 3124+4243=7367%3=2. That means we get base-9.

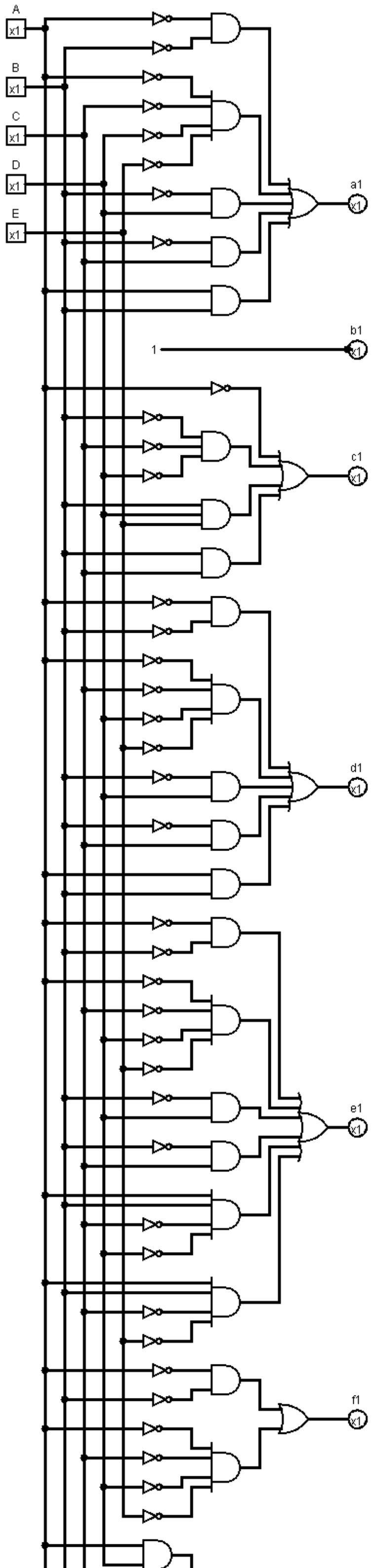
Again, we know that 5 bits is the maximum number of bits to be processed. Unlike the base conversion to octal, we can't separate the 2 LSB in the truth table but we can do it together. We need a truth table that has both the 1st and 2nd LSB of the output. Therefore, the truth table will have 32 rows and because the maximum value of the bit needed to be converted is 11110, it fits that we only need two 7-segment decoders for this and the rest will just become zeroes.

Here is the truth table:

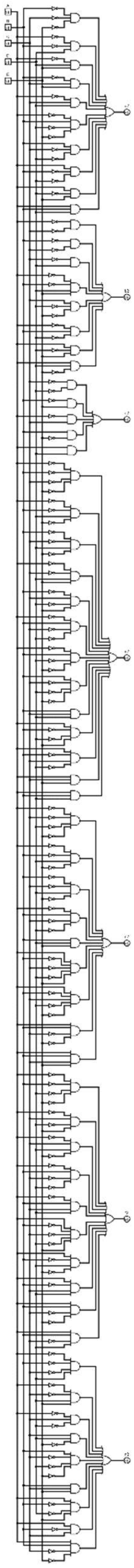


A	B	C	D	E	a1	b1	c1	d1	e1	f1	g1	a2	b2	c2	d2	e2	f2	g2	7-segment
0	0	0	0	0	1	1	1	1	1	1	0	1	1	1	1	1	1	0	"00"
0	0	0	0	1	1	1	1	1	1	1	0	0	1	1	0	0	0	0	"01"
0	0	0	1	0	1	1	1	1	1	1	0	1	1	0	1	1	0	1	"02"
0	0	0	1	1	1	1	1	1	1	1	0	1	1	1	1	0	0	1	"03"
0	0	1	0	0	1	1	1	1	1	1	0	0	1	1	0	0	1	1	"04"
0	0	1	0	1	1	1	1	1	1	1	0	1	0	1	1	0	1	1	"05"
0	0	1	1	0	1	1	1	1	1	1	0	1	0	1	1	1	1	1	"06"
0	0	1	1	1	1	1	1	1	1	1	0	1	1	1	0	0	0	0	"07"
0	1	0	0	0	1	1	1	1	1	1	0	1	1	1	1	1	1	1	"08"
0	1	0	0	1	0	1	1	0	0	0	0	1	1	1	1	1	1	0	"10"
0	1	0	1	0	0	1	1	0	0	0	0	0	1	1	0	0	0	0	"11"
0	1	0	1	1	0	1	1	0	0	0	0	1	1	0	1	1	0	1	"12"
0	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	1	"13"
0	1	1	0	1	0	1	1	0	0	0	0	0	1	1	0	0	1	1	"14"
0	1	1	1	0	0	1	1	0	0	0	0	1	0	1	1	0	1	1	"15"
0	1	1	1	1	0	1	1	0	0	0	0	1	0	1	1	1	1	1	"16"
1	0	0	0	0	0	1	1	0	0	0	0	1	1	1	0	0	0	0	"17"
1	0	0	0	1	0	1	1	0	0	0	0	1	1	1	1	1	1	1	"18"
1	0	0	1	0	1	1	0	1	1	0	1	1	1	1	1	1	1	0	"20"
1	0	0	1	1	1	1	0	1	1	0	1	0	1	1	0	0	0	0	"21"
1	0	1	0	0	1	1	0	1	1	0	1	1	1	0	1	1	0	1	"22"
1	0	1	0	1	1	1	0	1	1	0	1	1	1	1	1	0	0	1	"23"
1	0	1	1	0	1	1	0	1	1	0	1	0	1	1	0	0	1	1	"24"
1	0	1	1	1	1	1	0	1	1	0	1	1	0	1	1	0	1	1	"25"
1	1	0	0	0	1	1	0	1	1	0	1	1	0	1	1	1	1	1	"26"
1	1	0	0	1	1	1	0	1	1	0	1	1	1	1	0	0	0	0	"27"
1	1	0	1	0	1	1	0	1	1	0	1	1	1	1	1	1	1	1	"28"
1	1	0	1	1	1	1	1	1	0	0	1	1	1	1	1	1	1	0	"30"
1	1	1	0	0	1	1	1	1	0	0	1	0	1	1	0	0	0	0	"31"
1	1	1	0	1	1	1	1	1	0	0	1	1	1	0	1	1	0	1	"32"
1	1	1	1	0	1	1	1	1	0	0	1	1	1	1	1	0	0	1	"33"
1	1	1	1	1	x	x	x	x	x	x	x	x	x	x	x	x	x	x	

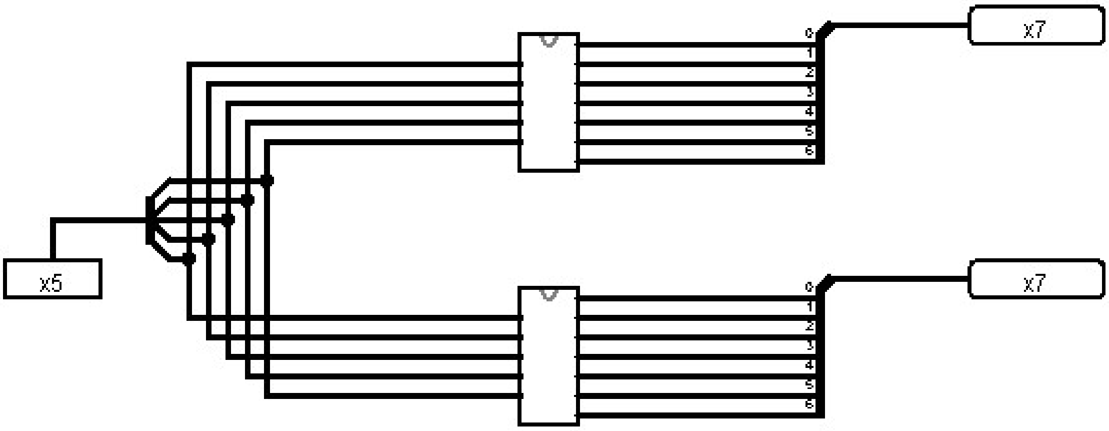
Because the use of Logisim's combinational analysis function is allowed, we did not used k-maps to form the circuit. We utilized build circuit of Logisim by inputting the truth table there and let it do the work for us. But Logisim can only analyze 12 outputs and we have 14 outputs. Note that the outputs of the 2nd LSB does not matter to the outputs of the 1st LSB and vice-versa. Thus, we created different sub circuits for those. The sub circuit for the 2nd LSB is here:


A
x1
B
x1
C
x1
D
x1
E
x1
a1
x1
1
b1
x1
c1
x1
d1
x1
e1
x1
f1
x1
g1
x1

This is the sub circuit for the 1st LSB:



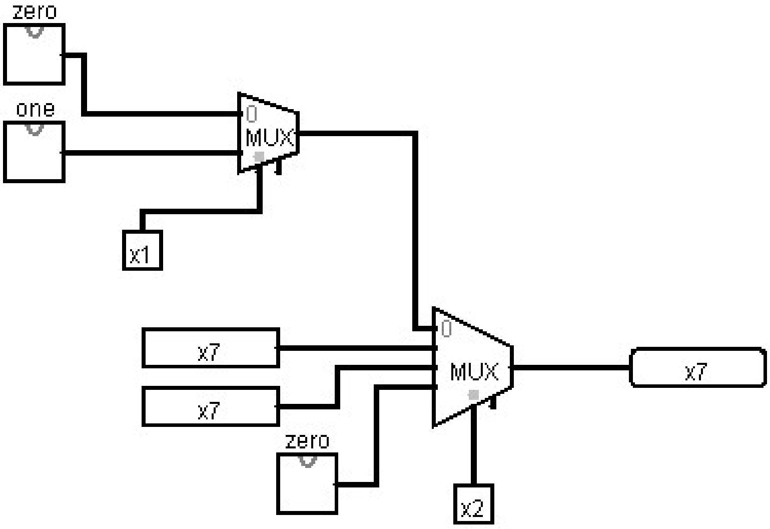
Like in the base conversion to octal, we can also create a sub circuit here that will join the 2 LSB to make it cleaner in the main circuit. The created sub circuit is below:



6. Base Selector to Output

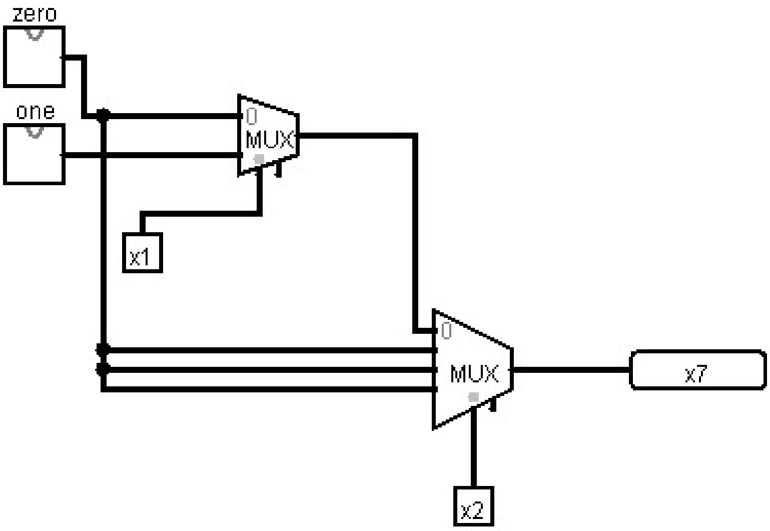
There are two choices we thought of. One is to analyze the base selector first and then, let only the bits to be converted to selected base but we don't know how to properly connect them to the output. That leads us to the second choice which is to simultaneously convert the bits to all the bases. We used a multiplexer to let the correct base be connected to the outputs.

We created two different sub circuits here just to make the main circuit simpler. The first one is the sub circuit for the 2 LSB outputs because the value of them changes mainly depending on the octal and base-9 conversion.



The topmost multiplexer has the inputs one and zero and as their name suggests, they are just 7-bit input that has the values of one and zero when put into the 7-segment decoders. That multiplexer is for processing the binary output of the circuit. It depends whether the current bit is 1 or 0. The 4-input multiplexer is the one that decides which value goes into the 7-segment decoders. The first input is for binary, octal for the second, base-9 for the third input, and zero for the last one. We just arbitrarily decided that the circuit will output all zeroes when the base selector is 11 instead of being valueless. Of course, it outputs binary when the base selector is 00, octal when 01, and base-9 when 10.

The second sub circuit is here:



This works almost the same as the previous sub circuit. The only difference is that we know that 3 of the bits will always be zero when octal or base-9 is selected. Therefore, we can just set them to zero.

For the remaining 2 MSB outputs, they are always zeroes no matter what the inputs are.

7. Combining the Circuits and Simplification

By combining the Operation sub circuit and the Base converter, we were able to create a functioning Adder/Comparator with Base Conversion which outputs into 7 7-Segment Displays. We managed to simplify the main circuit by making use of tunnels so that the wires look cleaner and make it easier to understand the whole circuit.

How the Whole Circuit Works:

The circuit works by having two 4-bit inputs, 1 2-bit input for the base conversion and 1 1-bit input. We have the 4-bit inputs showing its value through a hex digit display at the same time as being connected into connectors A and B respectively. We also take the connector, A/C, which is connected to the operations selector (0 – Adder, 1- Comparator). This connects the three inputs into the Operation sub circuit which returns a 5-bit binary value of the output. The output is then connected to the base converter which is determined by the 2-bit input and connected using the connector, base (00 – Binary, 01 – Octal, 10 – Base 9). The converted output then moves into splitters which enabled us to connect the output and display them into the 7 7-segment displays.