

Verilog toUpper

Kmap

The kmap was filled in using a python/pandas script in main.py. This script generates a csv file representation of the kmap where the lowercase letters are 1 while everything else is zero,

This was then imported into google sheets, where I highlighted the letters that would be changed by the function, and then mapped the kmap to be A5 out. I then imported an image of that into a pdf editing file and grouped my kmap as seen below.

A7A6A5A4\A3A2A1A0	0000	0001	0011	0010	0110	0111	0101	0100	1100	1101	1111	1110	1010	1011	1001	1000
0000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0001	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0011	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0010	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0110	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0111	0	0	0	0	0	0	0	1	1	1	1	0	1	0	0	0
0101	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1101	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1111	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1110	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1010	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1011	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1001	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

$$\begin{aligned}
 & A7A5 + A7'A6'A5 + A7'A5A4'A3'A2'A1'A0' + A7'A6A5A4A3A2 + A7'A6A5A4A3A2'A1'A0 \\
 & + A7'A6A5A4A3A2 + A7'A6A5A4A3A2'A1'A0
 \end{aligned}$$

The yellow background squares are the lowercase letters, which have their value forcibly set to one. Then I grouped my ones to get a Sum of Product equation:

$$F = A6'A5 + A7A5 + A7'A6A5A4'A3'A2'A1'A0' + A7'A6A5A4A3A2 + A7'A6A5A4A3A2'A1'A0$$

Testbench vvp

```

image-1.png image.png kmap_isLowercase_pandas.csv main.py README.md verilog
image-2.png kmap.csv kmap_solved.pdf pyproject.toml uv.lock

project1 on master is v0.1.0 via v3.13.5
> cd verilog

project1/verilog on master via V
> ls
aaron_james_toUpper_tb.v aaron_james_toUpper.v toUpper_tb.vcd toUpper.vvp

project1/verilog on master via V
> vvp toUpper
toUpper: Unable to open input file.

project1/verilog on master via V
> vvp toUpper.vvp
VCD info: dumpfile toUpper_tb.vcd opened for output.
time ns | in(dec) in(bin) in(char) | out(dec) out(bin) out(char)
-----|-----
25ns | 40 00101000 ( | 40 00101000 (
50ns | 72 01001000 H | 72 01001000 H
75ns | 183 10110111 . | 183 10110111 .
100ns | 131 10000011 . | 131 10000011 .
125ns | 124 01111100 | 124 01111100 |
150ns | 20 00010100 DC4 | 20 00010100 DC4
175ns | 235 11101011 . | 235 11101011 .
200ns | 97 01100001 a | 65 01000001 A
225ns | 65 01000001 A | 65 01000001 A
250ns | 122 01111010 z | 90 01011010 Z
275ns | 71 01000111 G | 71 01000111 G
300ns | 109 01101101 m | 77 01001101 M
325ns | 146 10010010 . | 146 10010010 .
350ns | 48 00110000 0 | 48 00110000 0
375ns | 207 11001111 . | 207 11001111 .
400ns | 58 00111010 : | 58 00111010 :
425ns | 123 01111011 { | 123 01111011 {
450ns | 144 10010100 . | 144 10010100 .
475ns | 127 01111111 DEL | 127 01111111 DEL
Test complete.
aaron_james_toUpper_tb.v:54: $finish called at 475019 (1ps)

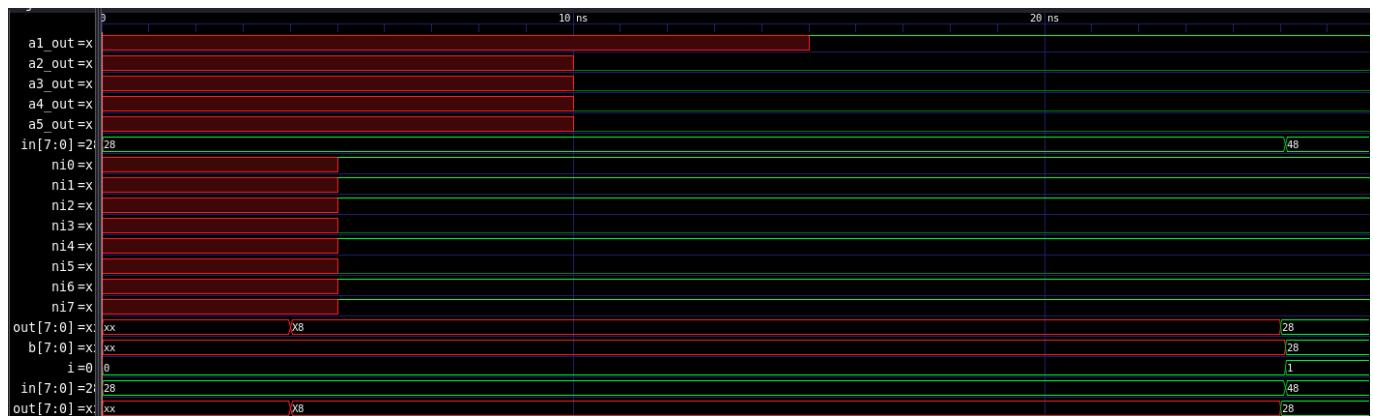
project1/verilog on master [!] via V
> []
[default] 0:bash*

```

"astral-debian" 10:01 04-Nov-25

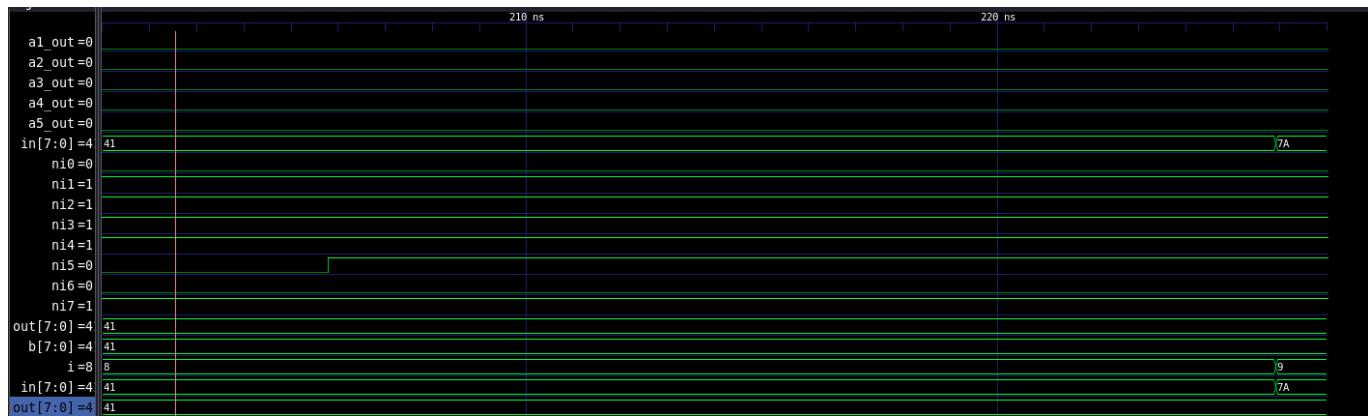
Here is the testbench vvp being executed after compiling. It shows that at any time higher then 25 ns, the circuit has correct output. The circuit is tested using 25.001 delay.

First waveform: '('



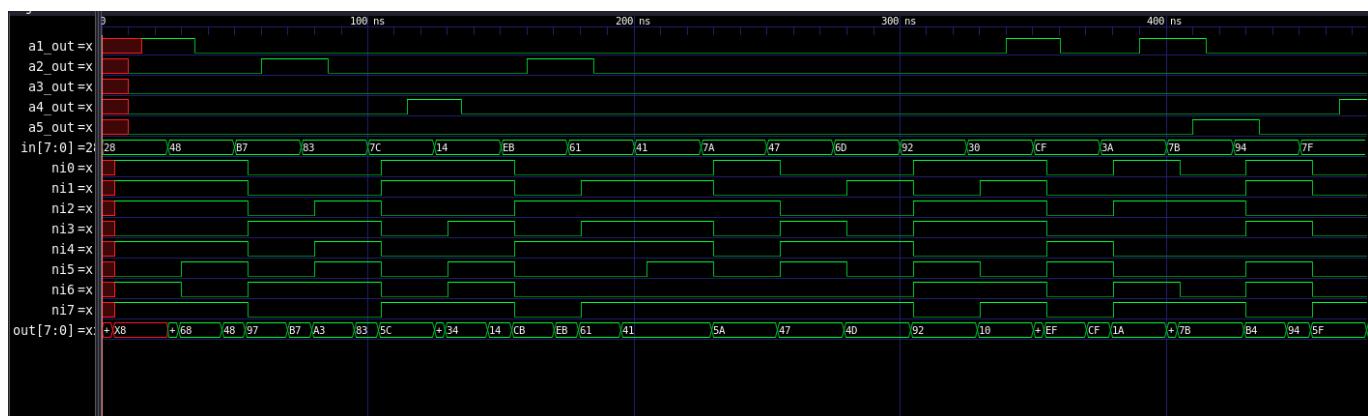
Here is the first test in waveform form, taking in '(' as an input. This returns '(' after 25 ns.

a -> A



Here is a later waveform, that takes in 'a' and returns 'A' after 25 ns.

full test bench



and here is the full testbench in waveform form, running all 19 tests with each taking 25 ns (with an extra .001 ns of delay.)