## Function Specification

This circuit is the Verilog implementation of the toUpper() function for an 8 bit ASCII input character. In the ASCII table, the characters for uppercase A through uppercase Z occupy the values 65–90, and the characters for lowercase a through lowercase z occupy the values 97–122. The difference between any lowercase alphabetical character and its corresponding uppercase character is always 32, or $2^5$. Thus, the fifth bit of the input determines whether the character is uppercase or lowercase. If the fifth bit is 0, it's an uppercase character, and if it's a 1 then its lowercase.

Therefore, to implement the toUpper() function, we must:

1. Detect when the input corresponds to a lowercase letter, which are ASCII values 97-122
2. Set bit 5 to 0 if it is lowercase
3. Leave all other bits unchanged

The function takes inputs 8 bits labeled A0-A7, and outputs 8 bits labeled B0-B7. Since we're only changing the 5th bit, B0, B1, B2, B3, B4, B6, and B7 are all identical to their input.

## Circuit Design

Since we do not have the luxury of something as simple as an `if` statement in Verilog, we must create a function to check whether the input is a lowercase letter. Below is the 8 variable k-map for the function L, which returns 1 if the input is a lowercase character and 0 otherwise:

| 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 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 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 |

In canonical minterm form, this is L = ∑m(97, 98, 99,...,122), or the sum of the 26 minterms representing a-z. Simplifying with the grouping shown above:

- red → A7A5
- blue → A7'A6'A5
- green → A7'A6A5A4A3A2
- orange → A7'A6A5A4A3A2'A1A0
- purple → A7'A5A4'A3'A2'A1'A0'

We notice that for all groupings, A5 is 1.Thus, the simplified boolean equation is

L = A5(A7 + A7’A6′ + A7’A6A4A3A2 + A7’A6A4A3A2’A1A0 + A7’A4’A3’A2’A1’A0’)

Now that we have a way to check if the input is a lowercase letter, we just have to change bit 5 if the input is not a lowercase letter:

$B5 = A5 * \bar{L}$

### Testbed Implementation

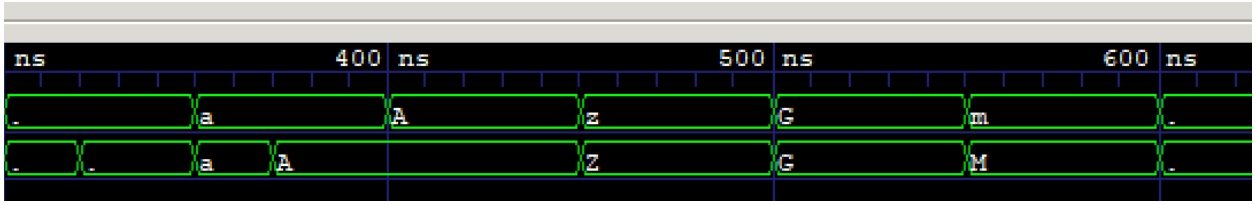
Here is the terminal output and waveforms on gtkwave with an initial input delay of 50 ns:

```
PS C:\Users\aslam\Desktop\softdev\cs211_project1> iverilog -o toUpper.vvp toUpper.v toUpper_tb.v
PS C:\Users\aslam\Desktop\softdev\cs211_project1> vvp .\toUpper.vvp
VCD info: dumpfile toUpper.vcd opened for output.

RESULT: 19 passed, 0 failed
toUpper_tb.v:60: $finish called at 950000 (1ps)
PS C:\Users\aslam\Desktop\softdev\cs211_project1> gtkwave .\toUpper.vcd

GTKWave Analyzer v3.3.100 (w)1999-2019 BSI

[0] start time.
[950000] end time.
```




Gradually decreasing the input delay, I settled on 26 ns as the minimum for the circuit to function properly.