Project 1 — Gate-level Implementation of toUpper() using Primitive Digital Gates

Name: Gaurav Banepali

Course: CSC 211000 – Digital Design

1. Purpose

The goal of this project was to design and test a digital hardware circuit that performs the 'toUpper()' function, which converts lowercase ASCII letters into their uppercase form. Unlike software-based implementations, this version uses only primitive logic gates to model the behavior at the hardware level. The project also explored how timing delays affect the accuracy and stability of the circuit when the input speed increases.

2. Design

The circuit was built in Verilog using only primitive gates such as NOT, AND, OR, NAND, NOR, XOR, XNOR, and BUF. Each gate type was assigned a fixed propagation delay to simulate real-world hardware timing. The logic works by clearing bit 5 (value 32) in the ASCII code for lowercase letters ('a' through 'z'), converting them to uppercase ('A' through 'Z'). For example, 'a' (01100001) becomes 'A' (01000001), and 'm' (01101101) becomes 'M' (01001101).

Gate Type	Delay (#)	Description
NOT	#5	Inverts a single bit
AND, OR	#10	Basic combinational gates
NAND, NOR	#12	Combined logic variations
XOR, XNOR	#15	Used for comparison and parity
BUF	#4	Buffers or stabilizes signal output

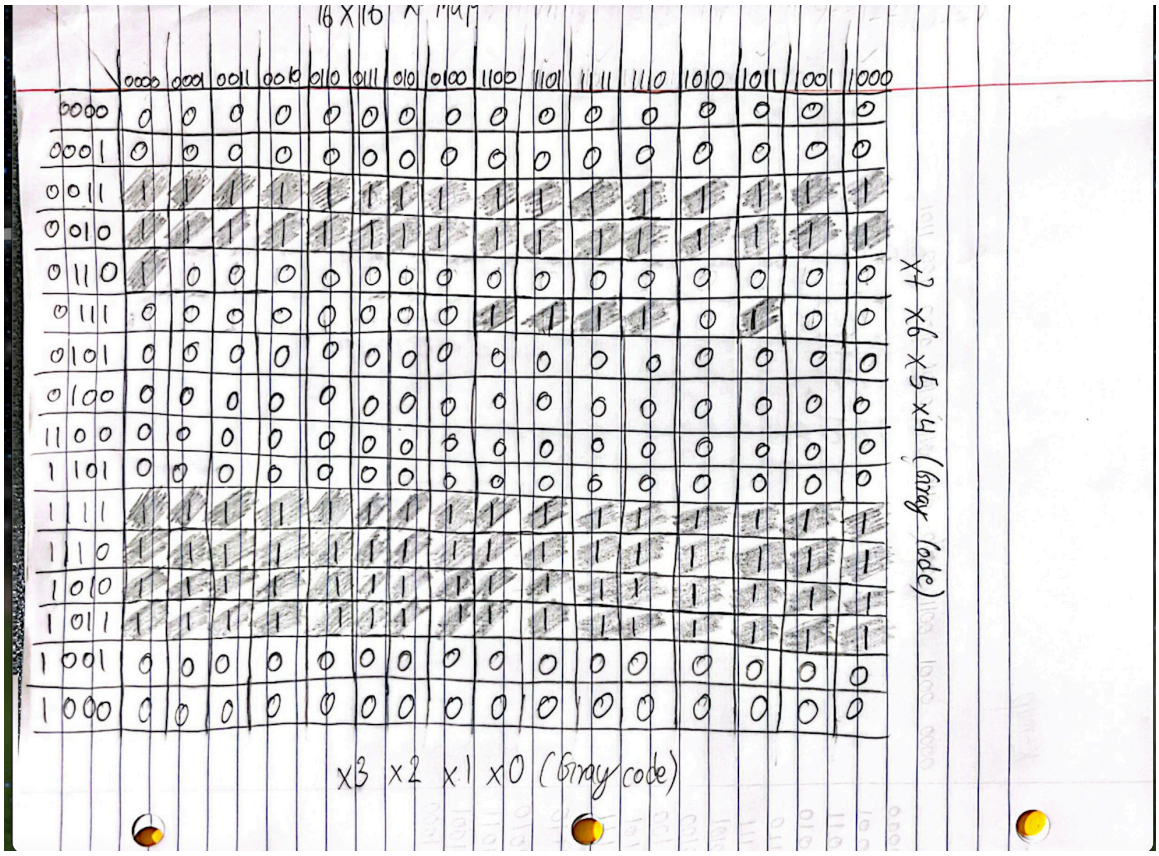
Two Verilog files were used for the implementation: 'toUpper_gates.v' contained the main circuit and 'tb_toUpper_gates.v' served as the testbench that provided inputs and recorded outputs. Simulations were run using the Icarus Verilog compiler, and waveform analysis was performed using GTKWave.

3.K-Map Analysis

To determine when the circuit should activate bit 5 (the uppercase conversion control), an 8-bit (16×16) K-map was constructed using Gray-code ordering for both the rows and columns.
The variable of interest is **y_5**, which corresponds to the fifth bit of the ASCII character code.
The logic was filled according to these rules:

- Output `0` when $x_5 = 0$.
- Output `1` when $x_5 = 1$, **except** for ASCII values 97–122 (`'a'` to `'z'`), where the cell is `0`.

This ensures that for lowercase letters (where bit 5 = 1), the circuit automatically clears that bit to convert the input to uppercase without any additional gate layer.
The simplified K-map expression obtained from this layout defines the minimal logic needed to drive the y_5 output using only primitive gates.



4. Simulation Results

The circuit was tested using several ASCII inputs including letters, digits, and symbols. At a 40 ns input interval, the system functioned perfectly. All lowercase letters were converted to uppercase while other characters remained unchanged.

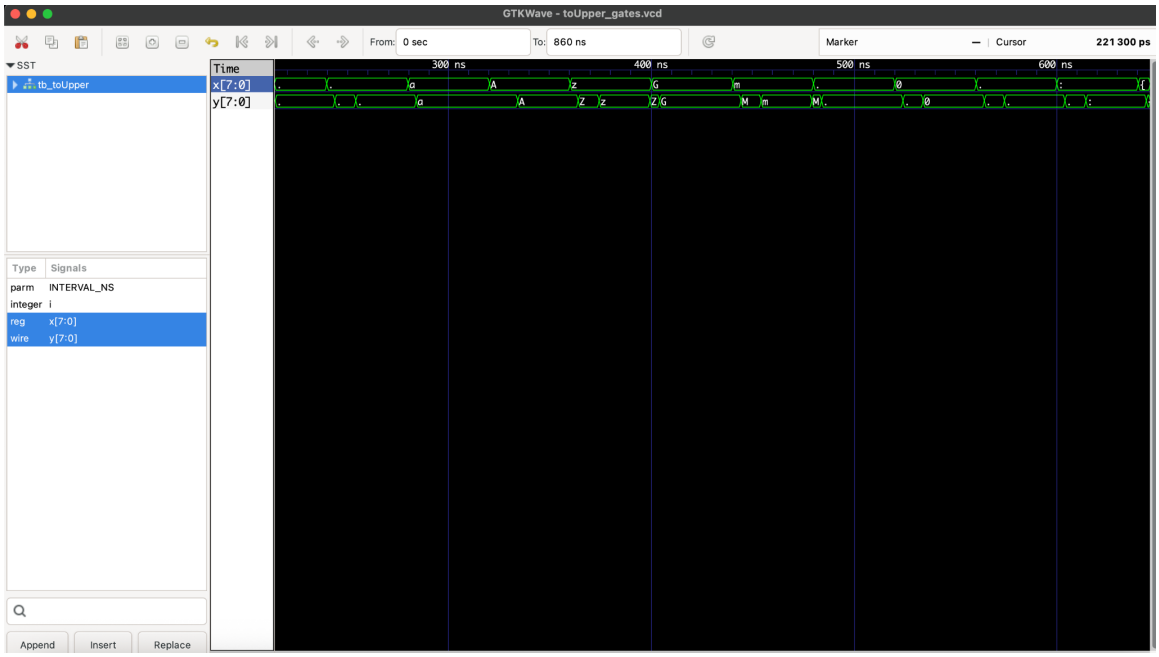


Figure 1: Normal operation at 40 ns showing proper lowercase to uppercase conversion.

5. Stress Testing and Timing Analysis

To find the speed limit of the design, the delay between inputs (INTERVAL_NS) was gradually reduced. When the interval was lowered below 10 ns, incorrect or unstable outputs started to appear. This indicates that 10 ns is the smallest safe delay that allows all logic gates to fully settle before the next input arrives.

Interval (ns)	Behavior
40	Correct behavior
20	Correct behavior
12	Correct behavior
10	✓ Minimum passing interval
8	✗ Failing interval (unstable output)

Below are the waveform images for the minimum passing and failing intervals:

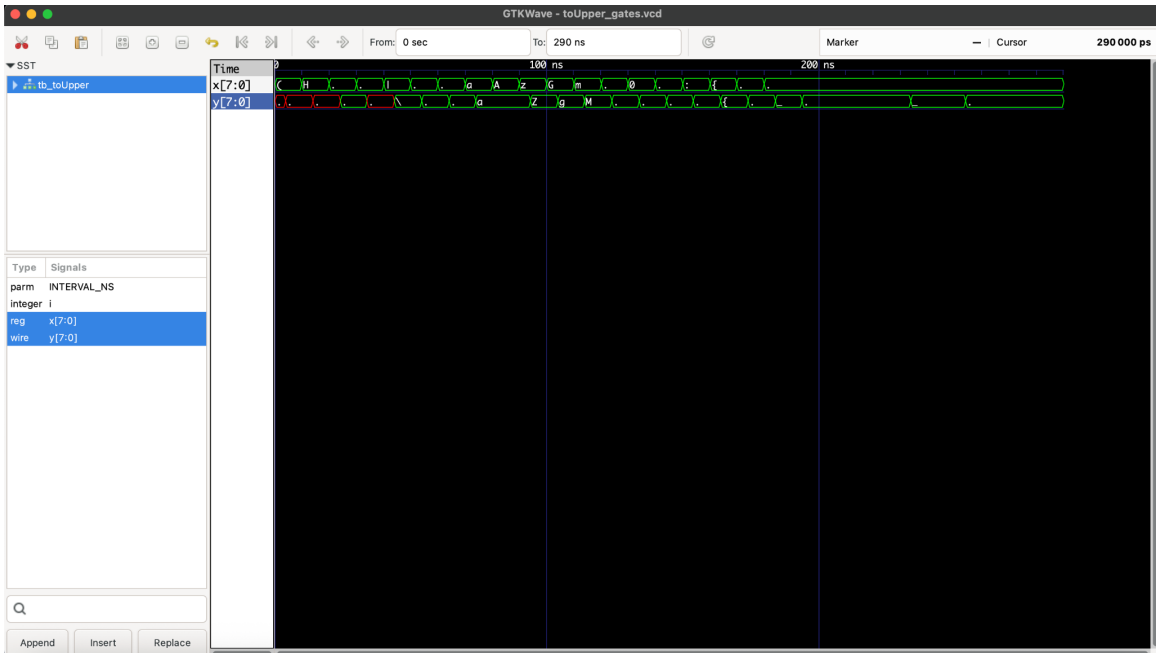


Figure 2: 10 ns — minimum delay with correct behavior.

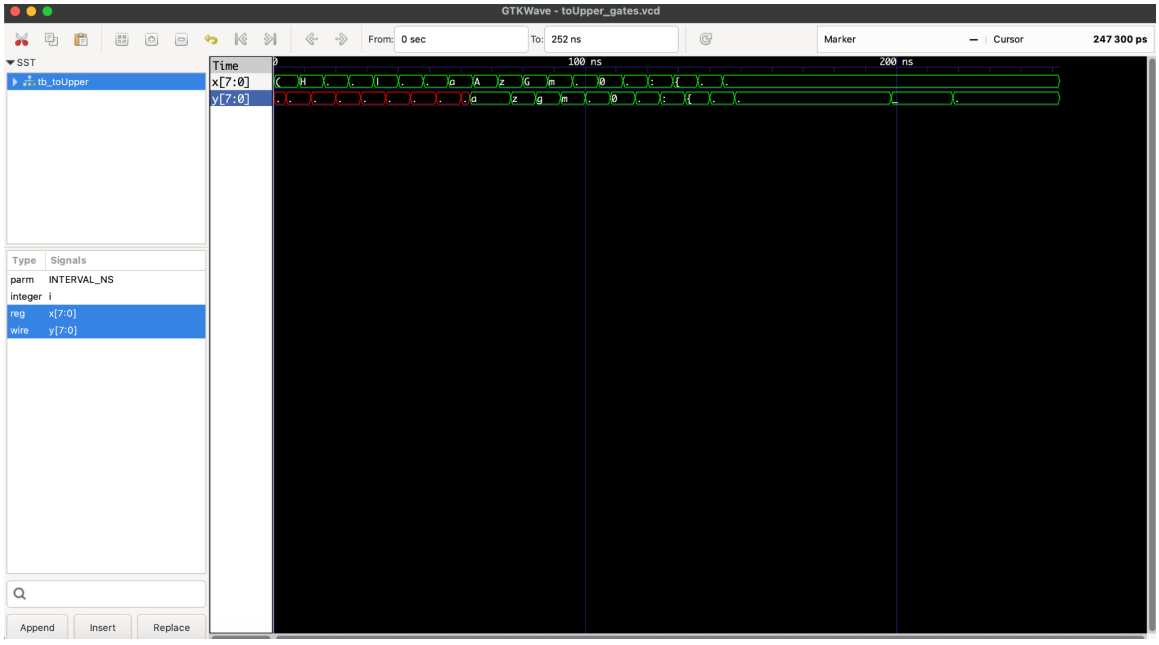


Figure 3: 8 ns — failing interval where output becomes unstable or incorrect.

6. Conclusion

The Verilog implementation of the 'toUpper()' function successfully demonstrates how logic gates can be used to mimic text manipulation at the hardware level. Through simulation and waveform analysis, it was found that the design works reliably at input delays of 10 ns or greater. Reducing the timing interval below this value leads to output errors, proving how

propagation delay affects circuit stability. This project highlights the connection between timing analysis and reliable hardware design.