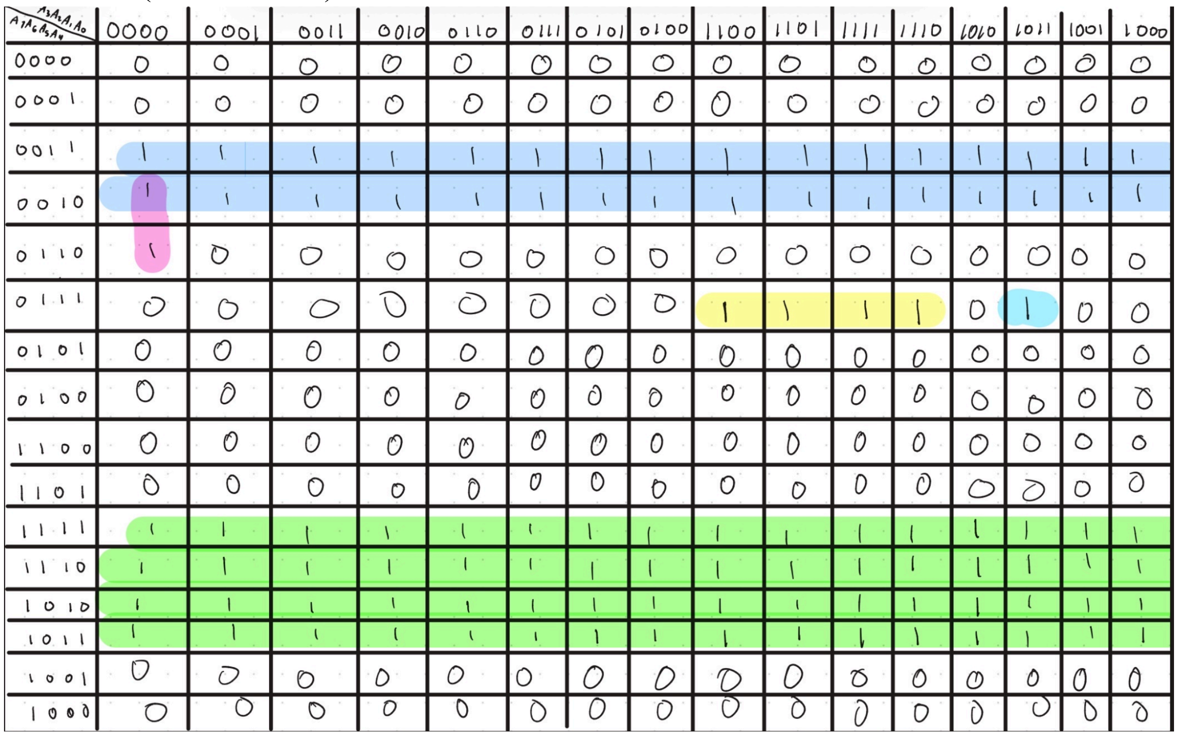
Raihan Nadeem Section CC3

Function specification: Describe in technology-agnostic language how the function operates. Note that this step should provide enough clarity such that someone unfamiliar with toUpper() could build the function correctly.

- The function, toUpper(), changes a lowercase letter into its uppercase form. It behaves similarly to the toUpper() function found in programming languages. For the Verilog implementation, the function takes an 8-bit ASCII input (a character) and then outputs an 8-bit value. The output is similar to the input except for the 5th bit. If the input character is a lowercase letter (a-z), the fifth bit is turned off (changed from 1 to 0) to produce its uppercase version. This is because in ASCII, the difference between lowercase and uppercase letters is bit 5, with uppercase letters having 1 for bit 5 and lowercase letters having 0 for bit 5. If the input is not a lowercase letter (anything besides a-z, such as numbers or already uppercase letters), the circuit outputs the same value without changing anything.

Circuit design:

- To create this circuit, I first began looking at the ASCII values represented in binary. For a-z, the values range from 01100001 (97) to 01111010 (122). For A-Z, the values range from 01000001 (65) to 01011010 (90). As discussed earlier, the only difference between lowercase and uppercase letters in ASCII is the 5th bit. Therefore, to design the circuit, it has to:
    - Read the input and detect if it is a lowercase letter
    - If the input is in the range from 01100001 to 01111010 (a lowercase letter), the 5th bit should be turned off (i.e., 1 → 0).
    - Otherwise, the bits should pass through as-is without any modifications.
- To create a Boolean expression that represents this design, I created an 8-variable Karnaugh map whose values represent the 5th bit. However, if it is a-z, the 5th bit is turned off (0 instead of 1).



$A_7A_6A_5A_4$ \ $A_3A_2A_1A_0$	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 represented as:

- F = $\sum m$(32 to 63, 96, 123 to 127, 160 to 191, 224 to 255), where F is the function represented by the Karnaugh map.

Simplifying this function using the Karnaugh map grouping:
BLUE: A7'A6'A5
PINK: A7'A5A4'A3'A2'A1'A0'
YELLOW: A7'A6A5A4A3A2
LIGHT BLUE: A7'A6A5A4A3A2'A1A0
GREEN: A7A5

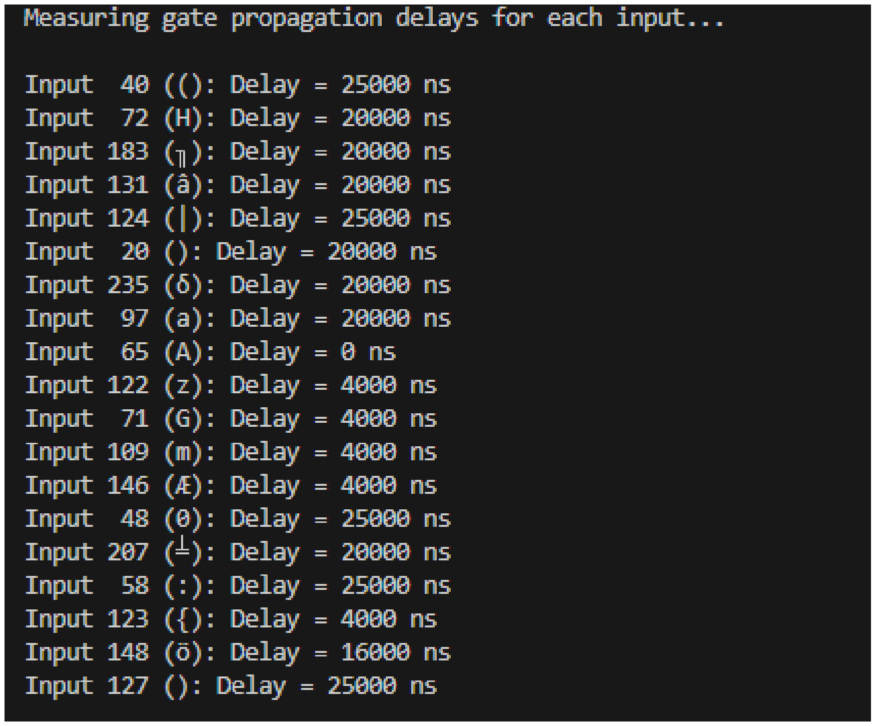
Summing these up, we have Sum of Products form: A7'A6'A5 + A7'A5A4'A3'A2'A1'A0' + A7'A6A5A4A3A2 + A7'A6A5A4A3A2'A1A0 + A7A5

This expression can be written in sum-of-products standard form if we were to distribute the terms, but for this Verilog implementation, I will leave it in this compact form, since we have already represented it in canonical minterm form.

Minimization Process: As shown above, the minimization for this expression was done through Karnaugh maps. Taking the canonical minterm form, I used an 8-variable Karnaugh map to derive the minimized SOP through 5 different groupings.

<u>Stress-testing and finding the smallest valid inter-input delay and the greatest invalid inter-input delay:</u>

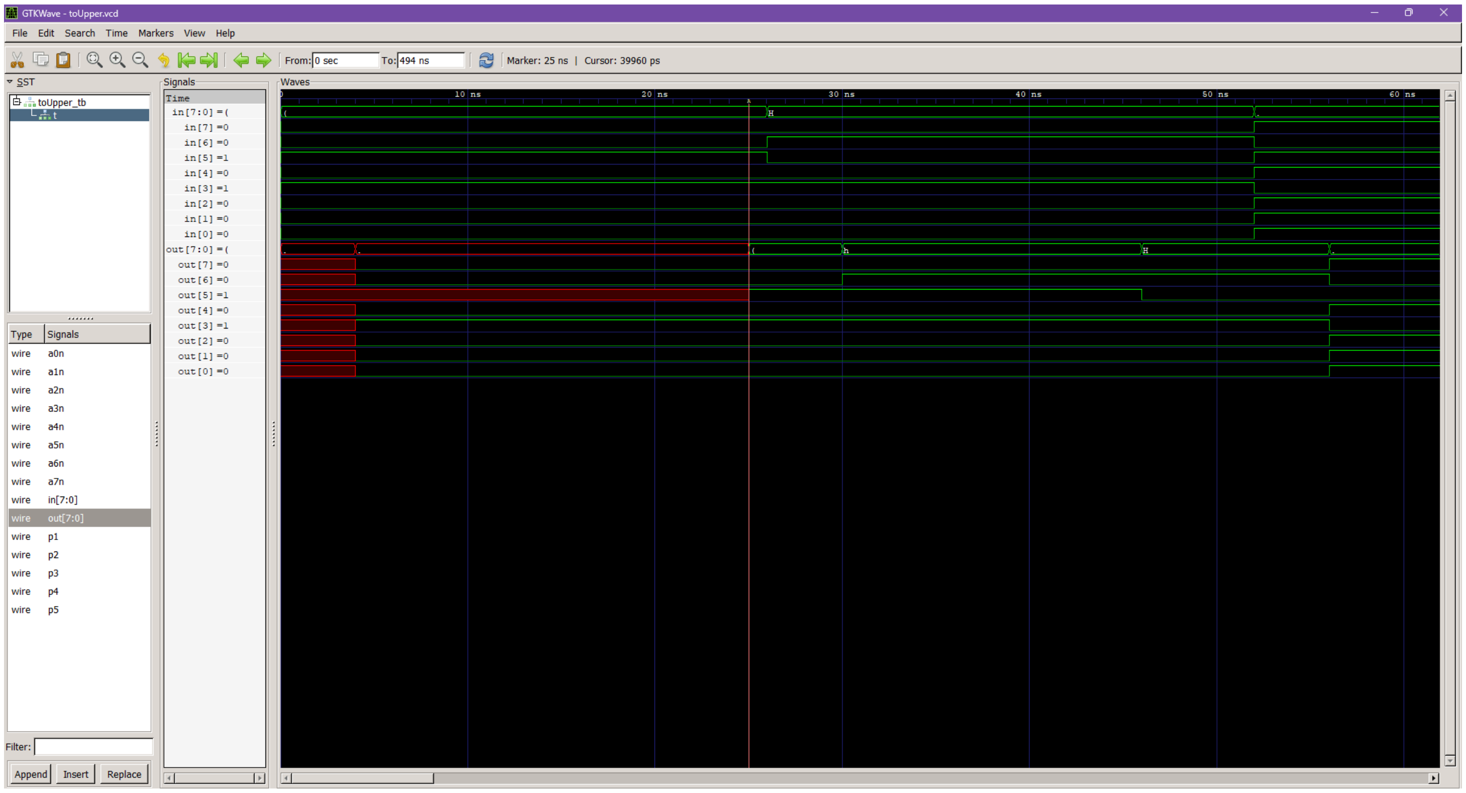
After implementing the Boolean expression into Verilog, I began to test my circuit. Since I have and gates, not gates, and or gates (and they run in parallel), it is expected that the circuit takes no more than 25 ns to complete based on the gate propagation delays. This was found to be true for all 19 test cases (delay here means how long the gate took, not the delay in between test cases):



```
Measuring gate propagation delays for each input...

Input  40 ((): Delay = 25000 ns
Input  72 (H): Delay = 20000 ns
Input 183 (╖): Delay = 20000 ns
Input 131 (â): Delay = 20000 ns
Input 124 (|): Delay = 25000 ns
Input  20 (): Delay = 20000 ns
Input 235 (δ): Delay = 20000 ns
Input  97 (a): Delay = 20000 ns
Input  65 (A): Delay = 0 ns
Input 122 (z): Delay = 4000 ns
Input  71 (G): Delay = 4000 ns
Input 109 (m): Delay = 4000 ns
Input 146 (Æ): Delay = 4000 ns
Input  48 (0): Delay = 25000 ns
Input 207 (╧): Delay = 20000 ns
Input  58 (:): Delay = 25000 ns
Input 123 ({): Delay = 4000 ns
Input 148 (ö): Delay = 16000 ns
Input 127 (): Delay = 25000 ns
```

To confirm, here is also a screenshot of my GTKWave with a marker at 25 nanoseconds, showing that the circuit completed within that timeframe: (better quality image here)



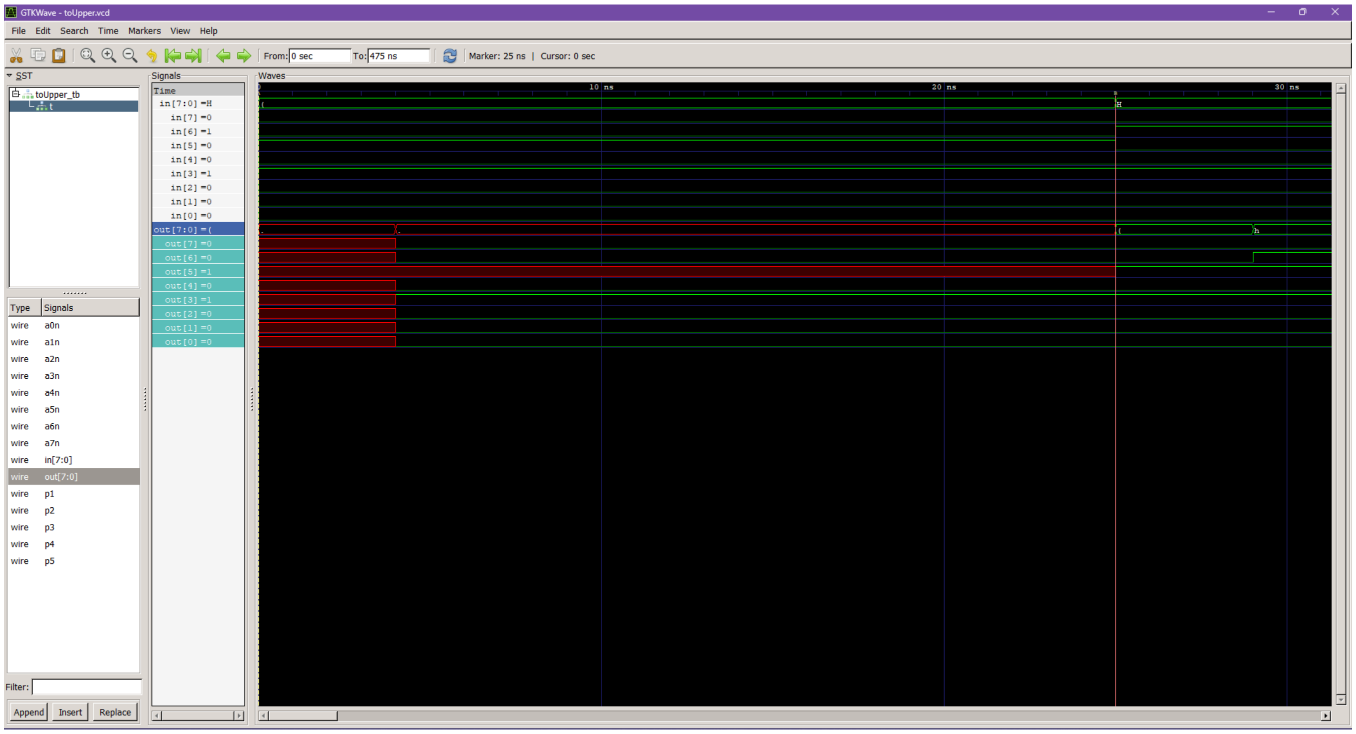
```
From: 0 sec   To: 494 ns   Marker: 25 ns | Cursor: 39960 ps

toUpper_tb
  t

Time
in[7:0] =(
  in[7] =0
  in[6] =0
  in[5] =1
  in[4] =0
  in[3] =1
  in[2] =0
  in[1] =0
  in[0] =0
out[7:0] =(
  out[7] =0
  out[6] =0
  out[5] =1
  out[4] =0
  out[3] =1
  out[2] =0
  out[1] =0
  out[0] =0

Type  Signals
wire  a0n
wire  a1n
wire  a2n
wire  a3n
wire  a4n
wire  a5n
wire  a6n
wire  a7n
wire  in[7:0]
wire  out[7:0]
wire  p1
wire  p2
wire  p3
wire  p4
wire  p5
```

However, something to note is that while the circuit may complete within 25 nanoseconds, it does not exhibit the correct behavior with a 25-nanosecond delay between each test case. From testing, I found that the **greatest invalid inter-input delay** is **25 nanoseconds,** where certain test cases fail. Below, you can see that at 25 nanoseconds, the circuit does not output the correct value: (better quality image here)



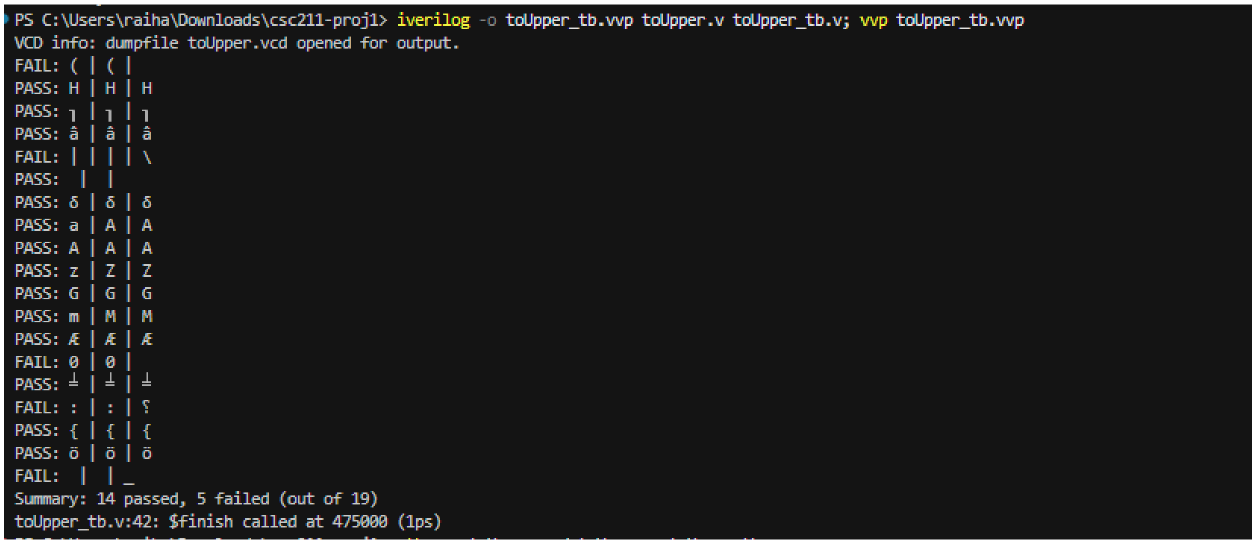
```
From: 0 sec   To: 475 ns   Marker: 25 ns | Cursor: 0 sec

toUpper_tb
  t

Time
in[7:0] =H
  in[7] =0
  in[6] =1
  in[5] =0
  in[4] =0
  in[3] =1
  in[2] =0
  in[1] =0
  in[0] =0
out[7:0] =(
  out[7] =0
  out[6] =0
  out[5] =1
  out[4] =0
  out[3] =1
  out[2] =0
  out[1] =0
  out[0] =0

Type  Signals
wire  a0n
wire  a1n
wire  a2n
wire  a3n
wire  a4n
wire  a5n
wire  a6n
wire  a7n
wire  in[7:0]
wire  out[7:0]
wire  p1
wire  p2
wire  p3
wire  p4
wire  p5
```

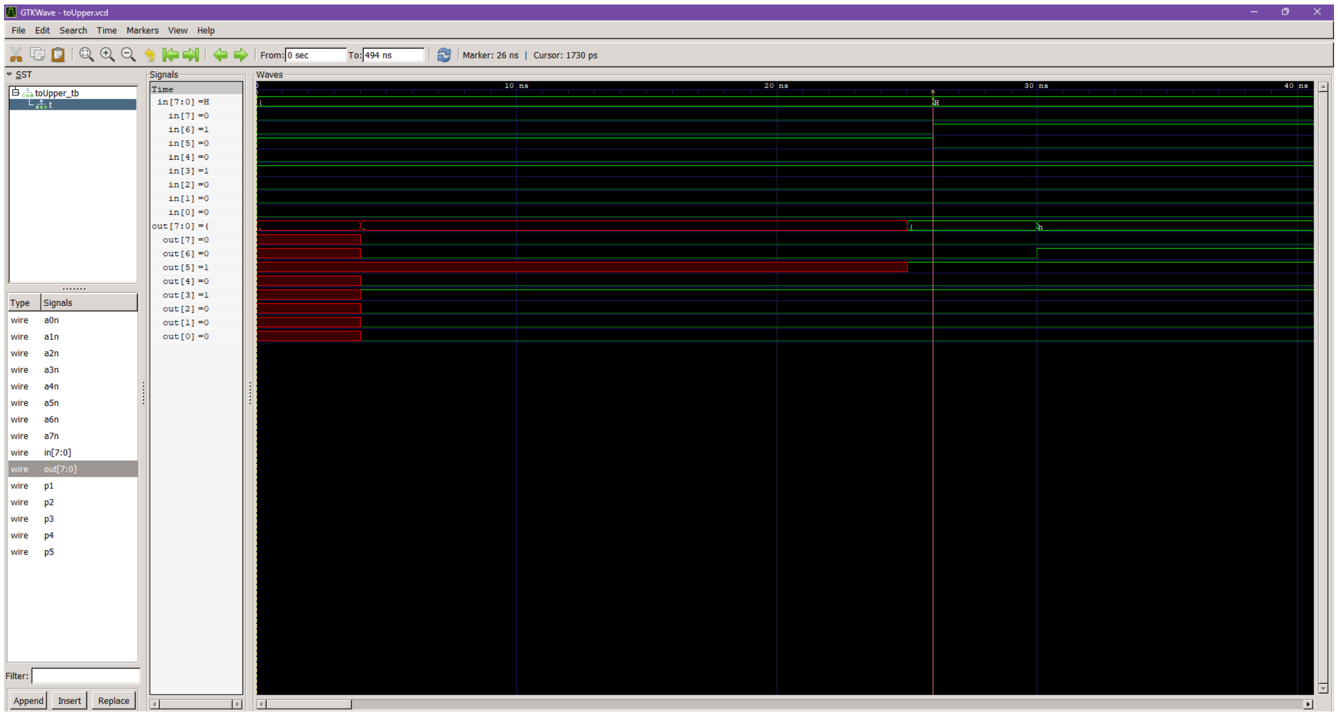
Console output at 25ns: (format: PASS/FAIL: Input | Expected | Output)



```
PS C:\Users\raiha\Downloads\csc211-proj1> iverilog -o toUpper_tb.vvp toUpper.v toUpper_tb.v; vvp toUpper_tb.vvp
VCD info: dumpfile toUpper.vcd opened for output.
FAIL: ( | ( |
PASS: H | H | H
PASS: ╕ | ╕ | ╕
PASS: â | â | â
FAIL: | | | | \
PASS:   |   |
PASS: δ | δ | δ
PASS: a | A | A
PASS: A | A | A
PASS: z | Z | Z
PASS: G | G | G
PASS: m | M | M
PASS: Æ | Æ | Æ
FAIL: 0 | 0 |
PASS: ╧ | ╧ | ╧
FAIL: : | : | ʕ
PASS: { | { | {
PASS: ö | ö | ö
FAIL:   |   | _
Summary: 14 passed, 5 failed (out of 19)
toUpper_tb.v:42: $finish called at 475000 (1ps)
```

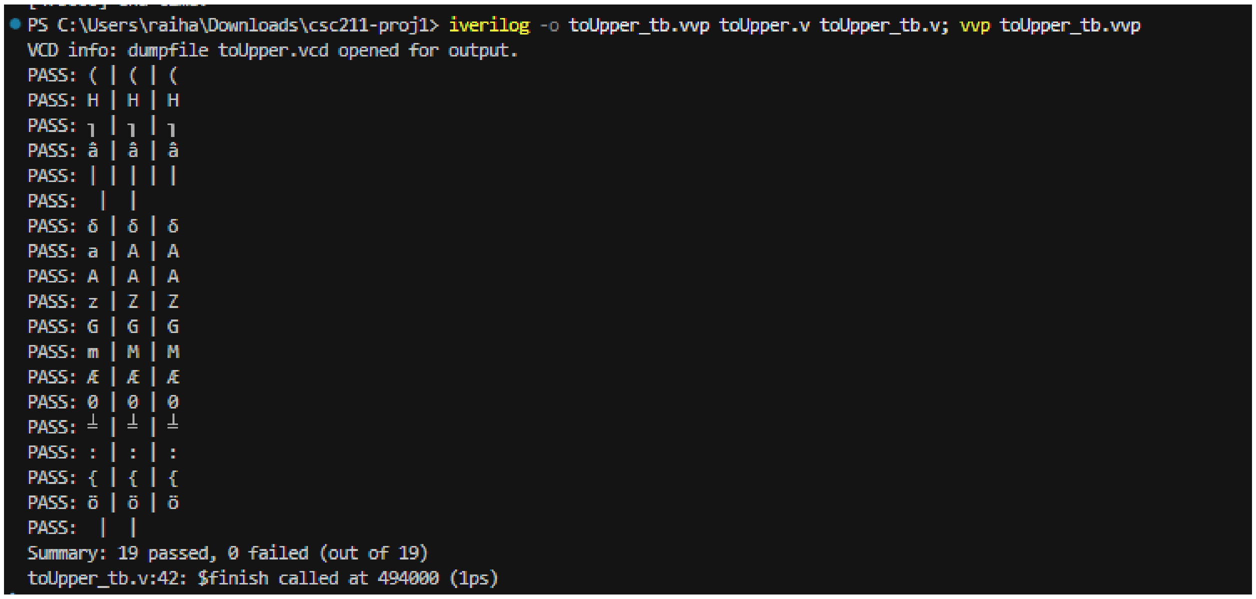
As you can see, 5 test cases failed at 25 ns, showing that it is not a long enough delay. This makes sense because the circuit takes 25 nanoseconds to complete, so there has to be a bigger delay.

After testing, I found the **smallest valid inter-input delay** to be **26 nanoseconds: (better quality image here)**



The circuit outputs the correct value with a 26-nanosecond delay, compared to a 25-nanosecond delay, where it does not output the correct value at 25ns, but rather a nanosecond later.

Console output with a 26-nanosecond delay: (format: PASS/FAIL: Input | Expected | Output)



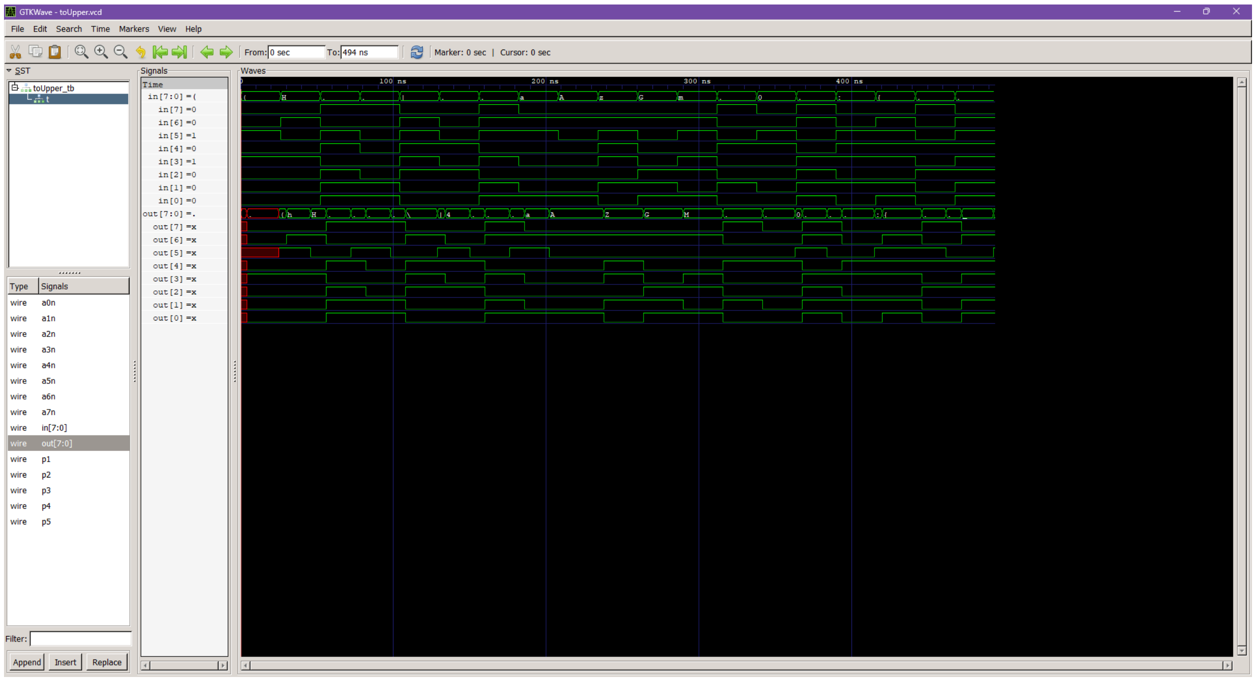
```
PS C:\Users\raiha\Downloads\csc211-proj1> iverilog -o toUpper_tb.vvp toUpper.v toUpper_tb.v; vvp toUpper_tb.vvp
VCD info: dumpfile toUpper.vcd opened for output.
PASS: ( | ( | (
PASS: H | H | H
PASS: ┐ | ┐ | ┐
PASS: â | â | â
PASS: | | | | |
PASS:   |   |
PASS: δ | δ | δ
PASS: a | A | A
PASS: A | A | A
PASS: z | Z | Z
PASS: G | G | G
PASS: m | M | M
PASS: Æ | Æ | Æ
PASS: 0 | 0 | 0
PASS: ╧ | ╧ | ╧
PASS: : | : | :
PASS: { | { | {
PASS: ö | ö | ö
PASS:   |   |
Summary: 19 passed, 0 failed (out of 19)
toUpper_tb.v:42: $finish called at 494000 (1ps)
```

As you can see, all test cases pass successfully at 26 nanoseconds with a total runtime of 494 nanoseconds (26 ns * 19 test cases = 494ns)

Therefore, the minimum delay such that the circuit exhibits correct behavior is at 26 nanoseconds.

Provide a screenshot of the waveform visualization of the output:

Here is an image of the output for all 19 test cases: (better quality image here)


GTKWave - toUpper.vcd
File Edit Search Time Markers View Help
From: 0 sec
To: 494 ns
Marker: 0 sec | Cursor: 0 sec
SST
toUpper_tb
t
Type Signals
wire a0n
wire a1n
wire a2n
wire a3n
wire a4n
wire a5n
wire a6n
wire a7n
wire in[7:0]
wire out[7:0]
wire p1
wire p2
wire p3
wire p4
wire p5
Filter:
Append Insert Replace
Signals
Time
in[7:0] =(
in[7] =0
in[6] =0
in[5] =1
in[4] =0
in[3] =1
in[2] =0
in[1] =0
in[0] =0
out[7:0] =.
out[7] =x
out[6] =x
out[5] =x
out[4] =x
out[3] =x
out[2] =x
out[1] =x
out[0] =x
Waves
100 ns
200 ns
300 ns
400 ns