Project 1 by Isaac Amos

Phase 1

Function specification:

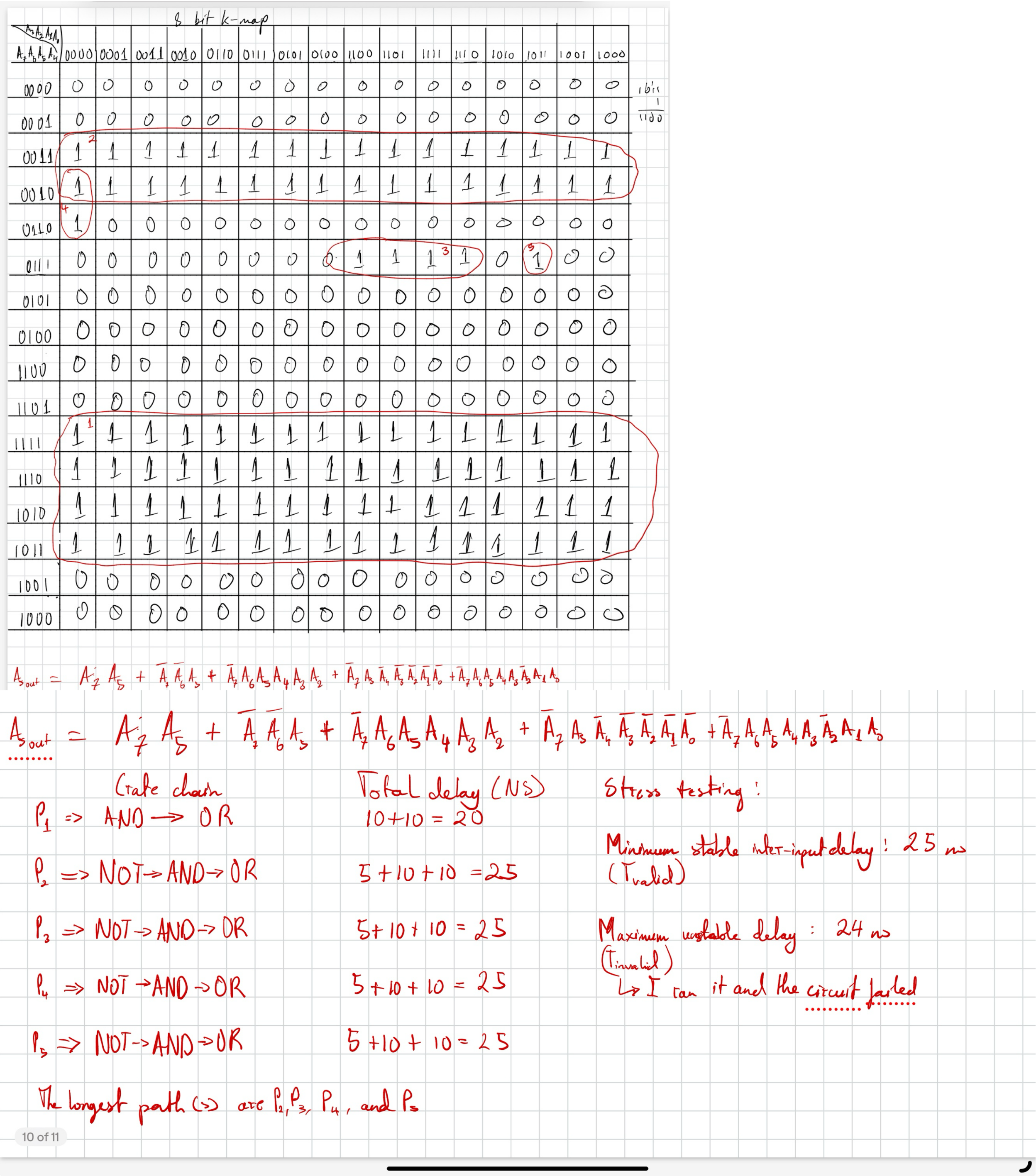
The toUpper() function is a combinational logic process designed to accept an 8-bit input character, represented by A in [7:0], and produce a modified 8-bit output character, A out [7:0]. The operation is entirely governed by the ASCII standard, which assigns specific numerical values to characters. The goal is to convert any lowercase English letter to its corresponding uppercase form, while leaving all other characters like , uppercase letters, numbers, symbols, and control codes, completely unaltered. The circuit must first identify whether the input falls into the narrow range of values that constitute a lowercase letter.

Furthermore, the conversion mechanism is confined entirely to the sixth bit, A5 (where A0 is the least significant bit). For all other seven bits, A out [i] where i is {0,1,2,3,4,6,7}, the output is simply connected directly to the input, such that A out [i] =A in [i]. The logic for the core bit is conditional, since all lowercase letters have A in [5] = 1, the conversion to uppercase requires this bit to be forced to 0 only when the L condition is active. If L is inactive, the output bit A out [5]must retain its original input value, A in [5]. This establishes the operational logic for the core bit as A out [5] = L AND A in [5], where L is the condition that the input is *not* a lowercase letter. This design strategy ensures a minimal circuit by isolating the complex logic to the generation of the L signal and the conditional manipulation of only one output bit.

Phase 2

Circuit design:

The circuit design phase details the unminimized (canonical) and minimized Sum-of-Products (SOP) Boolean expressions necessary for the implementation. For the seven non-critical output bits (A out[i] where i doesn't = 5), the canonical minterm form is the trivial identity function: A out[i] =A in[i]. The corresponding minimized SOP form remains A out[i] = A in[i], which translates directly to using a single Buffer (BUF) primitive gate for each of these seven outputs, ensuring minimal propagation delay for most of the circuit. The complexity is isolated to the sixth bit, A out[5], who's canonical minterm form is large, representing the sum of all 256 minterms where the input is *not* a lowercase letter, or where the input is a lowercase letter but the required output for Aout[5] is 0. This large canonical form must be minimized using Karnaugh maps to arrive at the efficient SOP form used for implementation. I did it in the screenshot below. The minimized SOP expression for A out[5], derived from the minimization process, is the final blueprint for this critical section of the circuit. The expression is: A out[5]= (A7A5) + (notA7 notA6 A5) + (notA7 A6 A5 A4 A3 A2) + (notA7 A5 notA4 notA3 notA2 not A1 notA0) + (notA7 A6 A5 A4 A3 notA2 A1 A0). I will implement it using the specified Verilog gate primitives (AND, OR, NOT) and their required propagation delays. This minimized form dictates the complete gate structure of the circuit and is the starting point for the Verilog implementation phase.



8 bit k-map

$A_7A_6A_5A_4$ \ $A_3A_2A_1A_0$	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

1 bit
1
1100

$A_{5out} = A_7 A_5 + \bar{A}_7 \bar{A}_6 A_5 + \bar{A}_7 A_6 A_5 A_4 A_3 A_2 + \bar{A}_7 A_5 \bar{A}_4 \bar{A}_3 \bar{A}_2 \bar{A}_1 \bar{A}_0 + \bar{A}_7 A_6 A_5 A_4 A_3 \bar{A}_2 A_1 A_0$

$A_{5out} = A_7 A_5 + \bar{A}_7 \bar{A}_6 A_5 + \bar{A}_7 A_6 A_5 A_4 A_3 A_2 + \bar{A}_7 A_5 \bar{A}_4 \bar{A}_3 \bar{A}_2 \bar{A}_1 \bar{A}_0 + \bar{A}_7 A_6 A_5 A_4 A_3 \bar{A}_2 A_1 A_0$

	Gate chain	Total delay (NS)
P_1	=> AND → OR	10+10 = 20
P_2	=> NOT→AND→OR	5+10+10 = 25
P_3	=> NOT→AND→OR	5+10+10 = 25
P_4	=> NOT→AND→OR	5+10+10 = 25
P_5	=> NOT→AND→OR	5+10+10 = 25

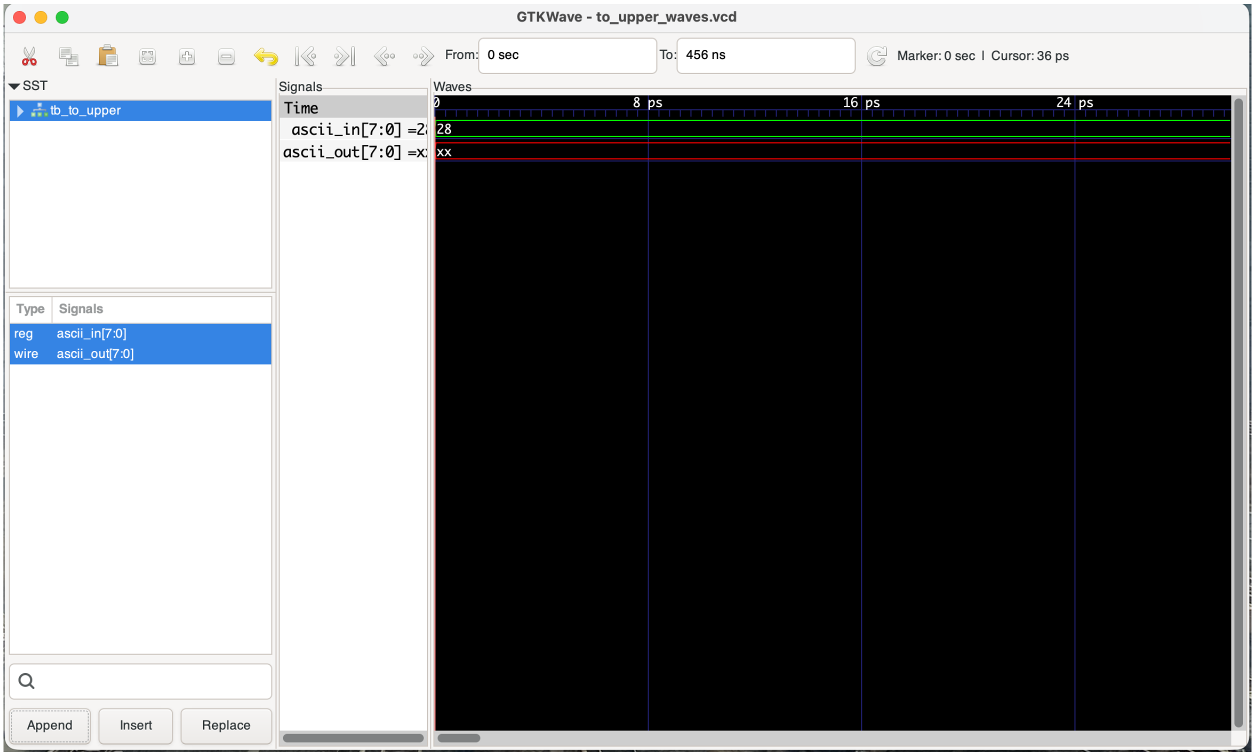
The longest path(s) are P_2, P_3, P_4, and P_5

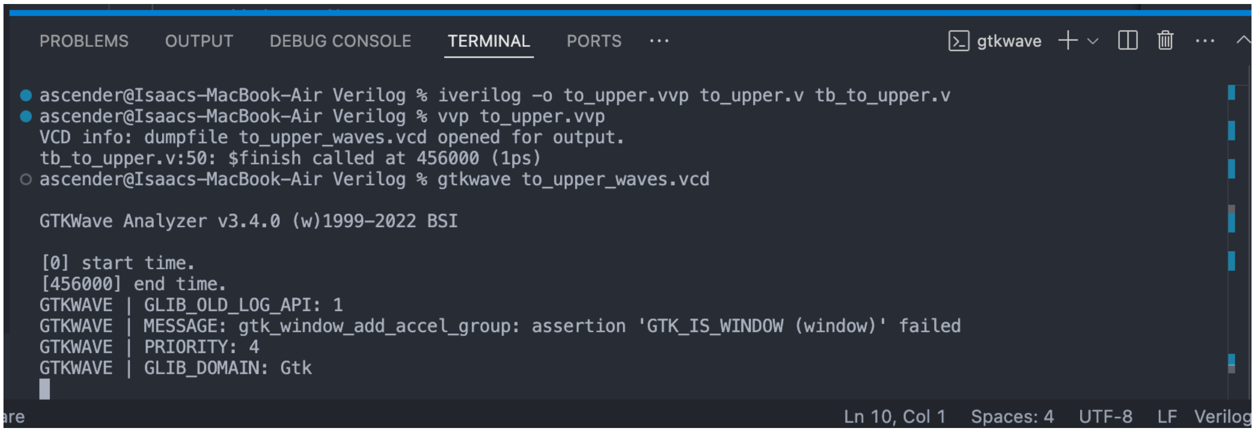
Stress testing:

Minimum stable inter-input delay: 25 ns
(T_{valid})

Maximum unstable delay: 24 ns
($T_{invalid}$)
↳ I ran it and the circuit failed

10 of 11

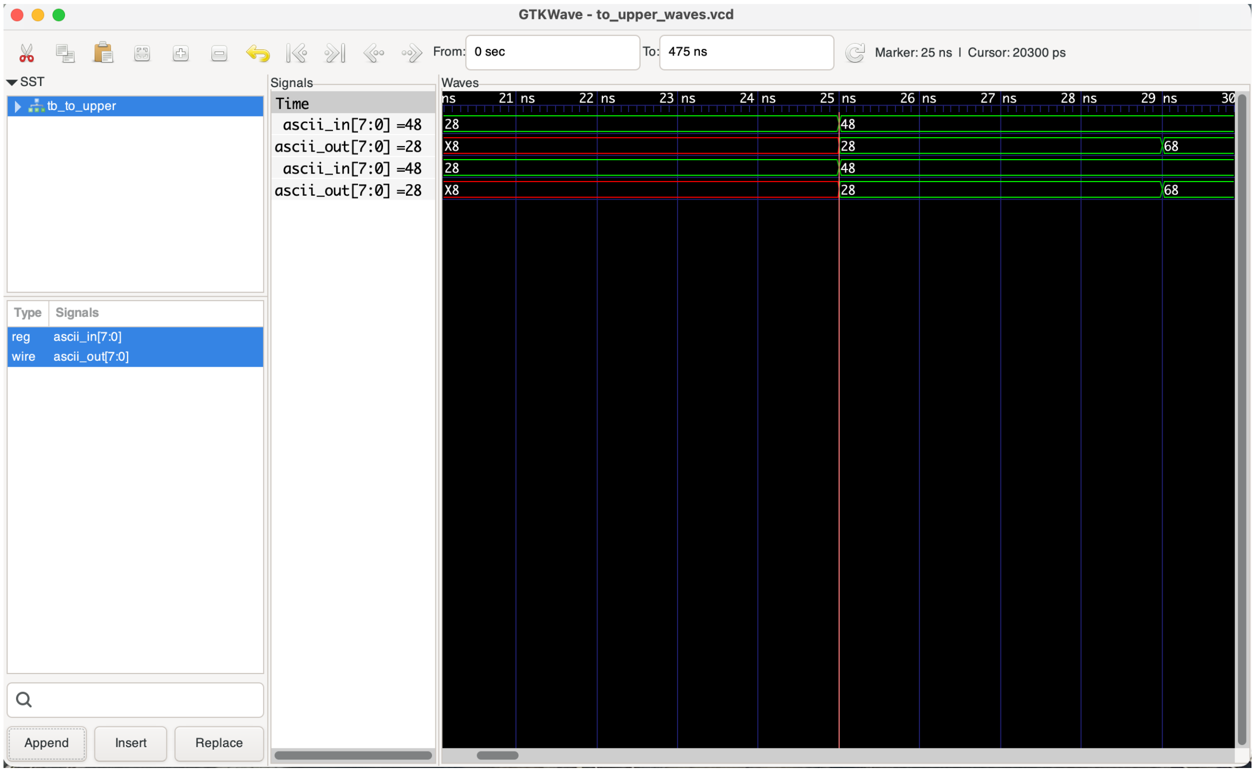




```
● ascender@Isaacs-MacBook-Air Verilog % iverilog -o to_upper.vvp to_upper.v tb_to_upper.v
● ascender@Isaacs-MacBook-Air Verilog % vvp to_upper.vvp
  VCD info: dumpfile to_upper_waves.vcd opened for output.
  tb_to_upper.v:50: $finish called at 456000 (1ps)
○ ascender@Isaacs-MacBook-Air Verilog % gtkwave to_upper_waves.vcd

  GTKWave Analyzer v3.4.0 (w)1999-2022 BSI

  [0] start time.
  [456000] end time.
  GTKWAVE | GLIB_OLD_LOG_API: 1
  GTKWAVE | MESSAGE: gtk_window_add_accel_group: assertion 'GTK_IS_WINDOW (window)' failed
  GTKWAVE | PRIORITY: 4
  GTKWAVE | GLIB_DOMAIN: Gtk
```



GTKWave - to_upper_waves.vcd

From: 0 sec To: 475 ns Marker: 25 ns | Cursor: 20300 ps

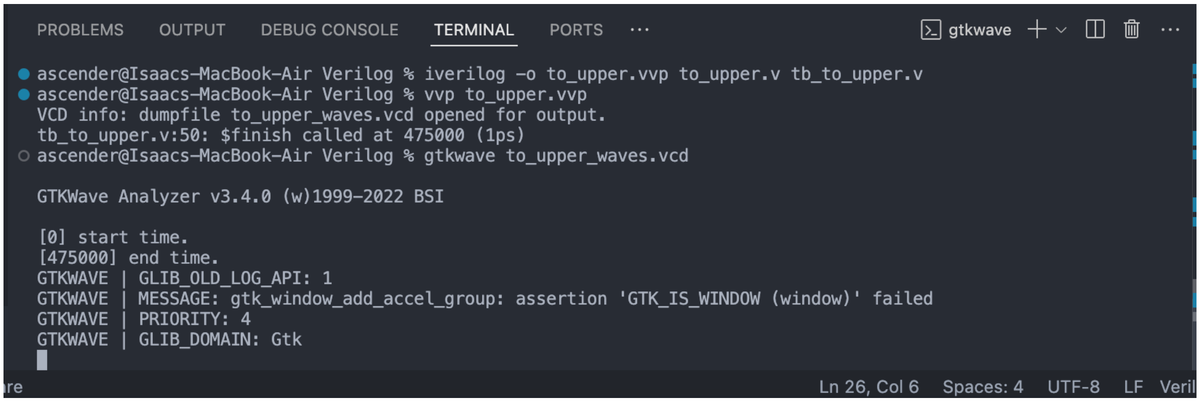
SST

tb_to_upper

Type	Signals
reg	ascii_in[7:0]
wire	ascii_out[7:0]

Signals

```
Time
 ascii_in[7:0] =48
ascii_out[7:0] =28
 ascii_in[7:0] =48
ascii_out[7:0] =28
```



```
● ascender@Isaacs-MacBook-Air Verilog % iverilog -o to_upper.vvp to_upper.v tb_to_upper.v
● ascender@Isaacs-MacBook-Air Verilog % vvp to_upper.vvp
  VCD info: dumpfile to_upper_waves.vcd opened for output.
  tb_to_upper.v:50: $finish called at 475000 (1ps)
○ ascender@Isaacs-MacBook-Air Verilog % gtkwave to_upper_waves.vcd

  GTKWave Analyzer v3.4.0 (w)1999-2022 BSI

  [0] start time.
  [475000] end time.
  GTKWAVE | GLIB_OLD_LOG_API: 1
  GTKWAVE | MESSAGE: gtk_window_add_accel_group: assertion 'GTK_IS_WINDOW (window)' failed
  GTKWAVE | PRIORITY: 4
  GTKWAVE | GLIB_DOMAIN: Gtk
```