Ambrose Wilkinson

Assignment 1:
to_upper() Function Specification

Function Specification

toUpper() is a commonly used programming function to convert ASCII character values from lowercase to uppercase. For example the string “tardy” would become “TARDY” after being passed through the toUpper() function. A char is made up of 8 bits of data ($b_7b_6b_5b_4b_3b_2b_1b_0$), from 01000001 - 01011010 for uppercase characters and 01100001 - 01111010 for lowercase characters.

One can assume that conversion of lower to upper relates only to the first four positions of the character, more specifically the third bit (b_2) and fourth bit (b_3) determine the casing of the letter due to the fact that letters change casing based on these bits alone.

As such, this assignment seeks to create toUpper() using binary logic and various gates. This will be represented using the HDL Verilog to run and test the logic as it would be used in an actual program.

Circuit Design

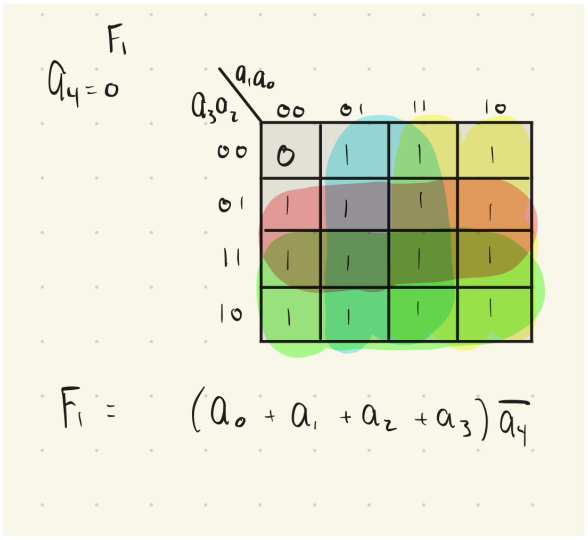
We start designing the circuit with a Kavnaugh map (K-map). Given that the function takes in 8 inputs and we know that upper and lower case characters of the ASCII tables only change by one bit, b_5, we write two K-maps. The first K-map looks at $b_4 = 0$ and maps $b_3b_2b_1b_0$ while the second K-map looks looks at $b_4 = 1$ and maps $b_3b_2b_1b_0$ (see figures below). Using these two K-maps we create a boolean expressions based on the locations of the maps that are true. These locations are the places where the function returns an ASCII character that is alpha, hence can be converted to uppercase if lower.

If this function returns true AND b_5 is 1, we know that the character inputted was alpha and lowercase. We, therefore, need to convert the input to an uppercase character. This is done by changing b_5 from 1 to 0 and returning the original inputs ($b_7b_6b_4b_3b_2b_1b_0$) unchanged. If, on the other hand, this function returns false, we know that the input is not alpha, and doesn’t need to be changed. In this case we will return the input unchanged.

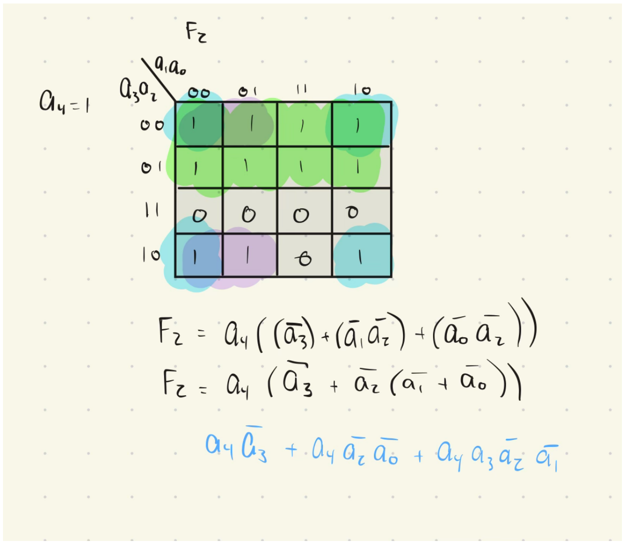
So I broke up the problem into three functions: the first handling the first 3 bits, $b_7b_6b_5$, the other two handing the last 4 bits $b_3b_2b_1b_0$ with the fifth bit, b_4 being toggled on and off. As such we arrived at $F1 = b_4'(b_3 + b_2 + b_1 + b_0)$, $F2 = b_4(b_3' + (b_1'b_2') + (b_1' b_0'))$, and $F3 = b_7'b_6b_5$. These functions and their k-maps may be seen below. From the derived k-maps a boolean gate function was created as can be seen below.

After writing this function in a verilog file, we test it in a verilog test bench and run it in GTKWave to see the time one the inputs and their time delays. The inputs used were: (, H, ·, *ƒ*, |, DC4, ë, a, A, z, G, m, ‘, O, Ï, :, {, “, and DEL. I ran into some issues because I wasn’t correctly stress-testing the propagation times. I started with 20 ns and didn’t realize that this would prevent my functions from working. After chatting with Professor Garrett, I discovered that these times were way too low and weren’t giving my function time to run. He suggested at larger time, 100ns and using binary search to discover the exact time when the function would work, which was 41ns. The results of this GTKWave form can be viewed below.

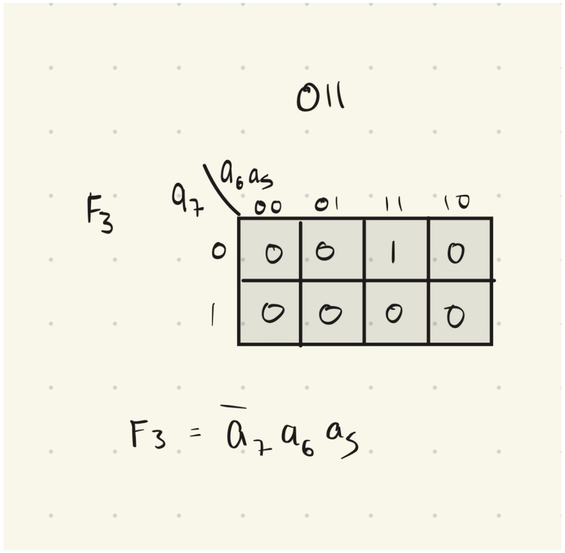
K-map of F1



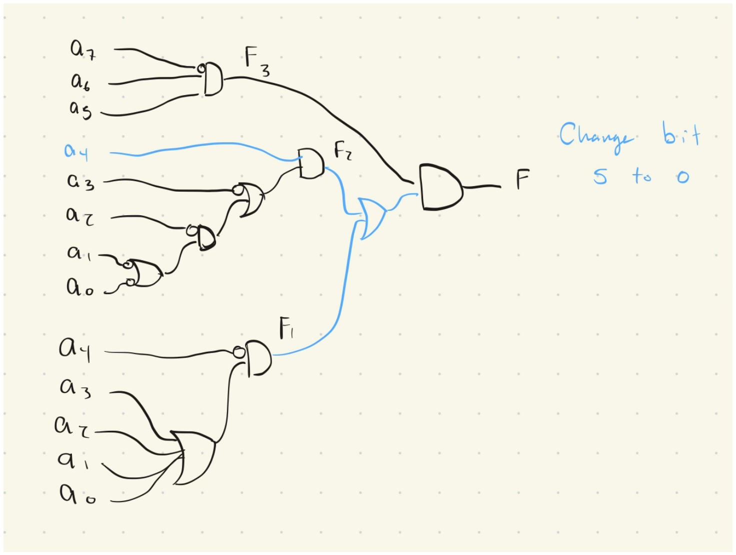
K-map of F2


F_2
$a_4 = 1$
$F_2 = a_4((\overline{a_3}) + (\overline{a_1}\,\overline{a_2}) + (\overline{a_0}\,\overline{a_2}))$
$F_2 = a_4(\overline{a_3} + \overline{a_2}(\overline{a_1} + \overline{a_0}))$
$a_4\overline{a_3} + a_4\overline{a_2}\,\overline{a_0} + a_4 a_3 \overline{a_2}\,\overline{a_1}$

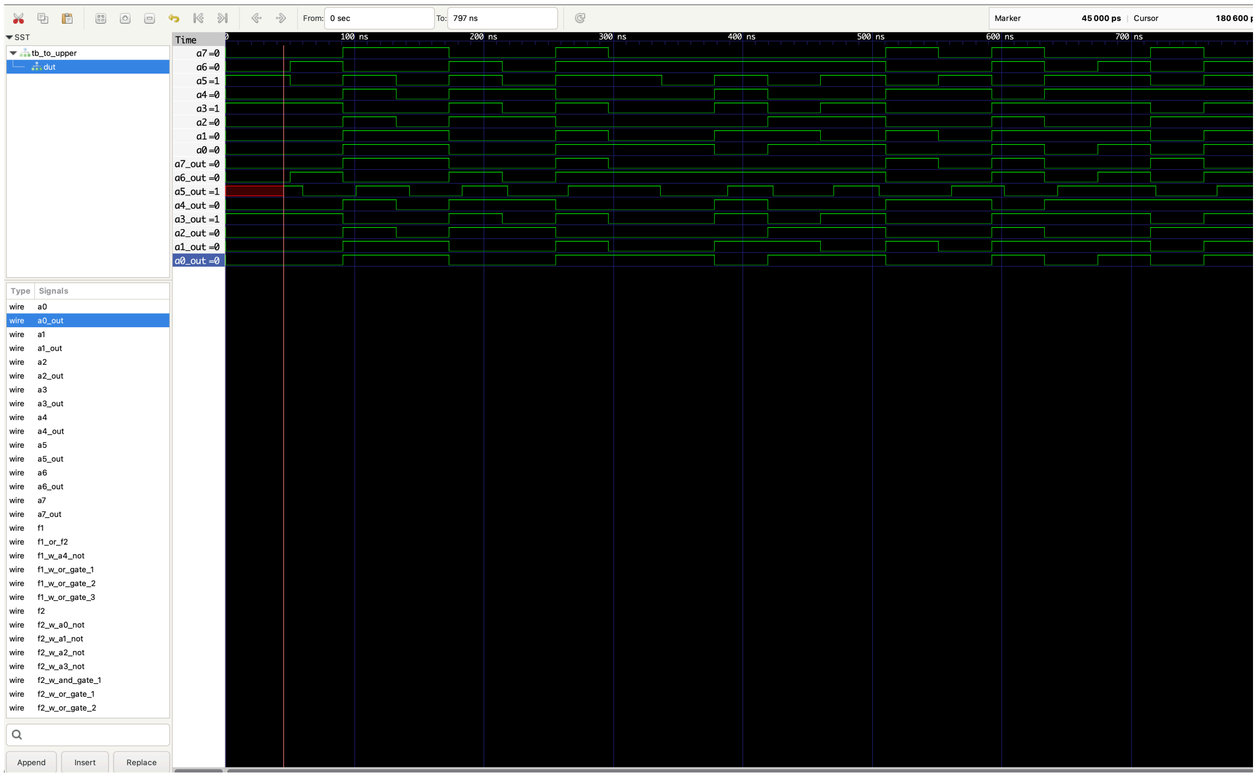
K-map of F3



Boolean Circuit



GTKWave Viewer



From: 0 sec To: 797 ns

Marker 45 000 ps | Cursor 180 600 p

SST
tb_to_upper
dut

Time
a7 =0
a6 =0
a5 =1
a4 =0
a3 =1
a2 =0
a1 =0
a0 =0
a7_out =0
a6_out =0
a5_out =1
a4_out =0
a3_out =1
a2_out =0
a1_out =0
a0_out =0

Type	Signals
wire	a0
wire	a0_out
wire	a1
wire	a1_out
wire	a2
wire	a2_out
wire	a3
wire	a3_out
wire	a4
wire	a4_out
wire	a5
wire	a5_out
wire	a6
wire	a6_out
wire	a7
wire	a7_out
wire	f1
wire	f1_or_f2
wire	f1_w_a4_not
wire	f1_w_or_gate_1
wire	f1_w_or_gate_2
wire	f1_w_or_gate_3
wire	f2
wire	f2_w_a0_not
wire	f2_w_a1_not
wire	f2_w_a2_not
wire	f2_w_a3_not
wire	f2_w_and_gate_1
wire	f2_w_or_gate_1
wire	f2_w_or_gate_2