Ramzey Ghanaim

Lab 3 Write-Up

April 22, 2016

Description

The purpose of this lab was to create a two's complement converter which implements an adder with multiplexers. The purpose of this lab is to become familiar with multiplexers and learn how I can implement them into digital logic designs. This lab required us to display an 8 bit number as two digits on the Basys2 Board's right two 7-segment LED displays. When the left most push button is pressed, the multiplexers will be enabled, allowing a 2's complement value of the original number to be calculated and displayed. For example, if the first switch on the board was enabled we would have the number 00000001, which will display 01 on the board. When the button is pushed, the two's complement will become 11111111 which will display FF on the board.

Methods

Adder

The first step in this lab was to construct the sign changer which will ultimately compute the two's complement (invert the bits and add one). I started by using two 4-1 multiplexers (muxes) to create the adder. One mux was used to calculate the sum of a single bit and the other was used to calculate the carry out. I created an adder in the last lab, however in order to replicate this process in a mux, I created two Karnaugh maps (K-maps) to help me organize the truth tables for the sum and carry of a single bit adder. From the K-map with the sum's values made four groups of conditions. For the four inputs of the mux. My four conditions are circled in the K-maps below.

Sum:

Cin / b / a	00	01	11	10
0	0	1	0	1
1	1	0	1	0

Table 1: K-map for the Sum of two bits and a carry

Cin:

a \ Cin b	00	01	11	10
0	0	0	1	0
1	0	1	1	1

Table 2: K-map for the Carry in of the sum of two bits

I then decided A will be the input variable while Cin and b would be the selectors. Cin would be s1 while b is s0. Also I circled the 4 groups of inputs I will have for each mux. The left most circle on both tables are the inputs for I_0 while the next circle from the left represents I_1. This pattern continues for both tables. Since A is the input variable, I had to come up with each input I_n (the Xilinx program refers to inputs with the letter D rather than I) that would fit with A. But since the first and third column both have all zeros and all ones, I simply said that I_0 and I_2 are just 1 and 0 respectively. The remaining two columns (second and fourth) contained the same values as A does in each row of these columns, and thus the input for the mux would just be A. Table 3 shows the truth table of the selectors, (C_{in} and B) and outputs for the sum (S) and carry out (C_{out}).

C_{in}	B		S	C_{out}
0	0		$D_0 = A$	$D_0 = 0$
0	1		$D_1 = \bar{A}$	$D_1 = A$
1	0		$D_2 = \bar{A}$	$D_2 = A$
1	1		$D_3 = A$	$D_3 = 1$

Table 3: Truth table of inputs for the S 4-1 mux and the C_{out} 4-1 mux

Once I put everything together, I came up with the result in figure 1:

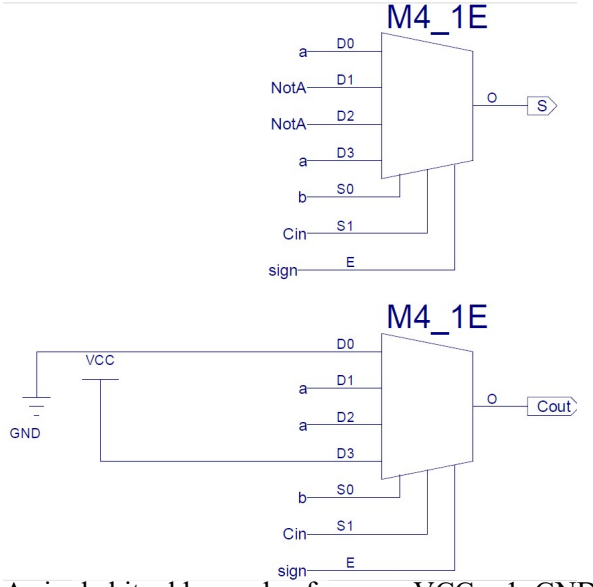


Figure 1: A single bit adder made of muxes. VCC = 1, GND = 0

One aspect of the muxes I implemented that I have yet to explain was the sign. The entire purpose of this adder is to add 1 when creating the 2's complement integer when the button is pressed. As a result, the sign is just the input for the button that will enable the muxes to compute the sum and carry out, as the same signal is connect to the enabler of both muxes.

Sign Changer

Once The single bit adder was done, I created a higher level schematic that would link 8 of the single bit adders together. This was the same process as in Lab 2. I connected the Cout of one adder to the Cin of the next. Also, the inputs of the two bits being added together came from the input of the switches, and 1, as the whole point of the adder is just to add 1 to create the 2's complement number. Once again, to create a 1 I used Vcc, and to create 0 I used GND. Once the adders were configured, I had to create muxs for each bit to select which output is to be displayed: the original number when sign is 0 (button is not being pressed) or the 2's complement when sign = 1 (button is being pressed). Once this was done I had to create input markers for the sign and each bit from the switches. I immediately inverted these bits, as the 2's complement requires an inversion of each bit. I then sent the inverse of each bit to the adders. My final schematic can be seen in the Appendix, as it would we too large to recreate as a figure.

7-Segment Display

Once I finished my schematic, I began to work on the 7-segment display. To start, I replicated my truth table I made for the last lab's 7-segment display. And just like the adder, I created K-maps, but this time the K-maps were for each LED on the display (CA – CG). The K-maps can be seen in tables 4-9.

n_1n_0 \ n_3n_2	00	01	11	10
00	0	1	0	0
01	1	0	1	0
11	0	0	0	1
10	0	0	0	0

Table 4: K-map for the CA

n_1n_0 \ n_3n_2	00	01	11	10
00	0	0	1	0
01	0	1	0	0
11	0	0	1	1
10	0	1	1	0

Table 5: K-map for CB

n_1n_0 \ n_3n_2	00	01	11	10
00	0	0	1	0
01	0	0	0	0
11	0	0	1	0
10	1	0	1	0

Table 5: K-map for CC

n_1n_0 \ n_3n_2	00	01	11	10
00	0	1	0	0
01	1	0	0	1
11	0	1	1	0
10	0	0	0	1

Table 6: K-map for CD

n_1n_0 \ n_3n_2	00	01	11	10
00	0	1	0	0
01	1	1	0	1
11	1	1	0	0
10	0	0	0	0

Table 7: K-map for CE

n_1n_0 \ n_3n_2	00	01	11	10
00	0	0	0	0
01	1	0	1	0
11	1	0	0	0
10	1	1	0	0

Table 8: K-map for CF

n_1n_0 \ n_3n_2	00	01	11	10
00	1	0	1	0
01	1	0	0	0
11	0	1	0	0
10	0	0	0	0

Table 9: K-map for CG

Once The K-maps were created, I divided up each map into 8 pieces for each of the 8 inputs in the mux. I used n_0 as the input variable while n_1-n_3 were used as selectors. Once again all these muxes had enabler signals, which comes in handy when selecting which digit to create for the display. I had a total of 7 muxes, and one inverter for inputs that required n_0 . once this schematic was complete, I proceeded to create the top level schematic.

Top Schematic

To begin the top level schematic, I crated I/O BUFs for the switches and button 0 (IBUFs) and 7-segment LED's (OBUFs). Once all my BUFs were in order I added a sign changer symbol and two 7-segment display (hex7seg) symbols. I needed two hex7seg symbols because we needed 2 hex digits to represent the 8 binary switches. One symbol was used for each digit, AN0 and AN1. Next I had to add a signal called dig_sel. The Basys2 Board can only power one digit at a time on the board. As a result the dig_sel signal was created to alternate between the two digits. If dig_sel was 1 it would power the first digit, AN0. And if dig_sel was 0, it would power the second digit, AN1. I had dig_sel connect directly to the enable of the muxes in the 7-segment display muxes. Once the signal was connected, I then downloaded the provided Verilog source code "digsel.v" and added it to my project. This is verlog code which controls the dig_sel signal. The next step was to create a symbol for digsel, and connect it to the proper IBUFG location, B8. Now that the digit selector was complete it was time to simulate the design.

Simulator

Now that the project was complete, it was time to test it in the simulator. A new "Verilog Text Fixture" was added to the project. I then copied the provided code that represents the input for digit selector. This code simulates the digsel symbol in my schematic, and constantly changes the dig_sel at a high frequency (more on this in the results section). Once I added the provided code I then began adding my own code, that turned on and disabled switches to display every character (0-F) on each digit. Switches 0-3 controlled the first digit while switches 4-7 were used to control the second digit. I turned these switches on and off to stimulate each character on both LEDs for 100 ns. Each character was displayed on both digits at the same time. Once I verified my simulator was operating as I wanted, turning on the appropriate LEDs, the next step was to stimulate the button for some of the characters. Once again I verified the simulator was operating as I had intended before I went on to add tests for different characters at each digit. I tested the following characters: df, 42 , 69, and da. The simulator was ran once again to verify the appropriate LED's were lighting up. Lastly I added code to simulate the button during for these last four tests. During the tests, the appropriate 2's complement value was displayed.

Results

After the success of the simulator I implemented my design on the Basys2 Board. The board worked as desired, displaying the appropriate numbers on the display as well as the appropriate 2's complement number when the button was pushed. Using my knowledge from Lab 1, I connected the board I was working on to the oscilloscope to measure the frequency of dig_sel, to see how fast the digits were switching on the 7-segment display. I had to edit my schematic to send dig_sel to pin JB1before measurements could be taken. Once that was complete, the graph in Figure 2 was displayed.

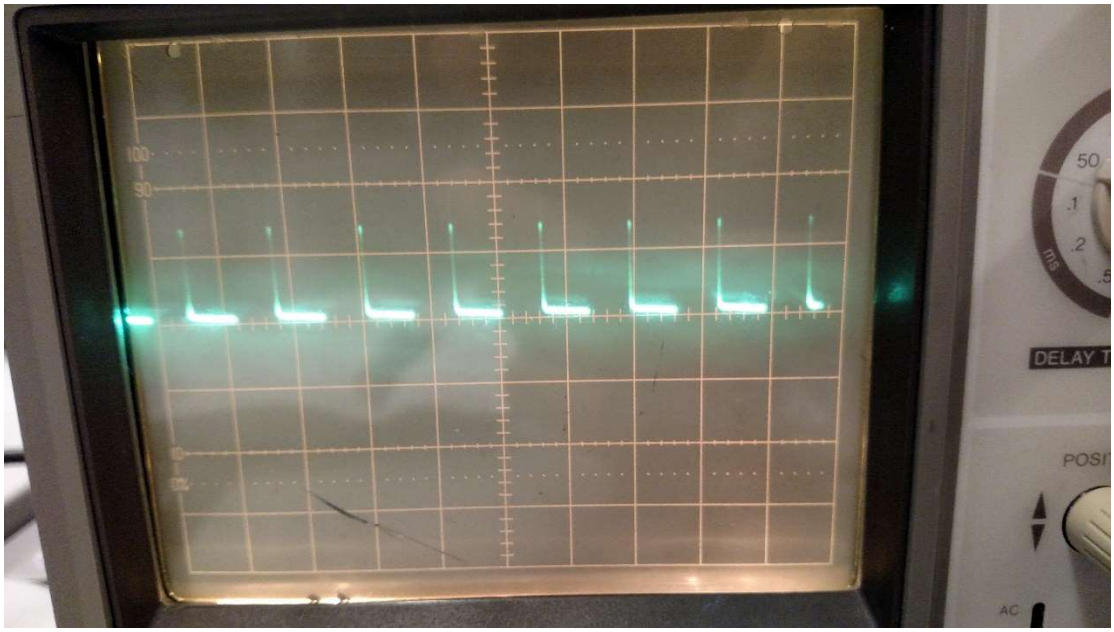


Figure 2: Graph of dig_sel's signal oscillation

The distance of one bar on this graph was measured to last for 1.3 ms. The period is 1.3 x 10^{-3}.

Dig_sel is oscillating at a frequency of $\frac{1}{1.3\times10^{-3}}$ = 769.2 Hz.

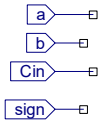
With a frequency this high, I was unable to view flickering of the 7-segment display. Also there seems to be a small glitch on the left side of each dash on the graph shown in Figure 2. This is most likely a result of human error from the connections between the board an oscilloscope.

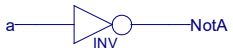
Conclusion

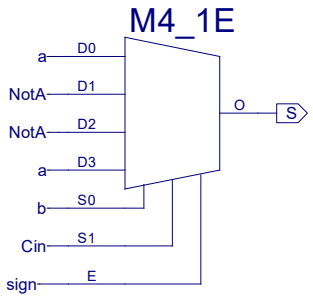
In this lab I learned how to used multiplexers to create adders, how to use multiple digits on the 7-segment display through alternation of power to each digit, and I became more familiar with the simulator and programing it with Verilog code. One challenge I faced was getting the simulator to work once I wrote in my tests in Verilog. It took me a while to find out that I did not initialize all the switches and buttons being used to 0. As a result of this error, none of the LEDs (CA-CG) were on. I also had a lot of trouble to get the still graph on oscilloscope. It took a lot of time playing around with the horizontal sweep and trig level, moving the nobs in slight increments to achieve a still graph. If I could do this lab again, I would try to understand Verilog code syntax and style before I start programming. Looking through my design, there are not any components I would optimize as the K-maps helped me find optimal solutions.

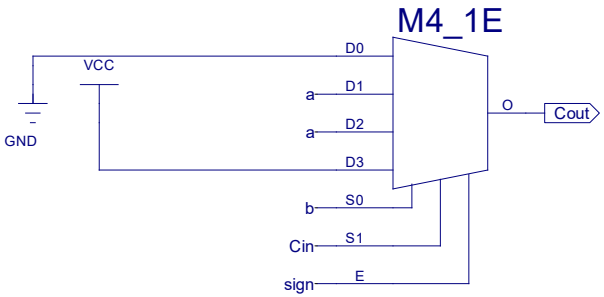
Appendices on next page

Adder

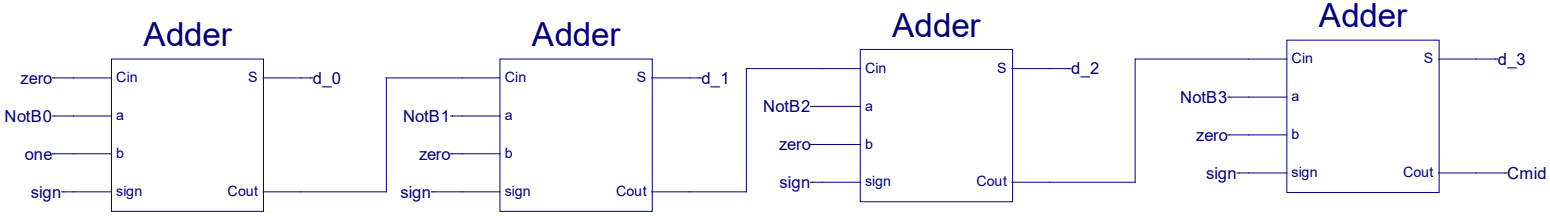

a
b
Cin
sign

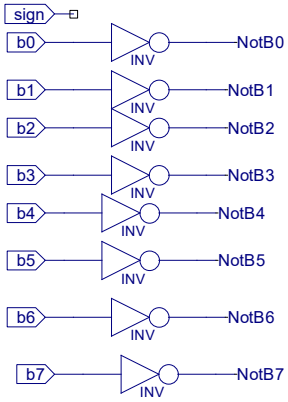

a
NotA
INV

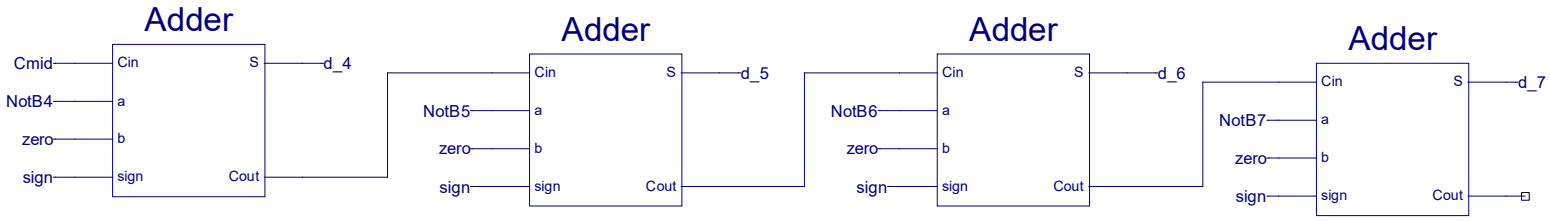

M4_1E
a
NotA
NotA
a
b
Cin
sign
D0
D1
D2
D3
S0
S1
E
O
S

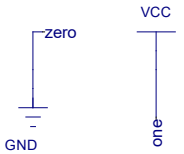

M4_1E
VCC
GND
a
a
b
Cin
sign
D0
D1
D2
D3
S0
S1
E
O
Cout

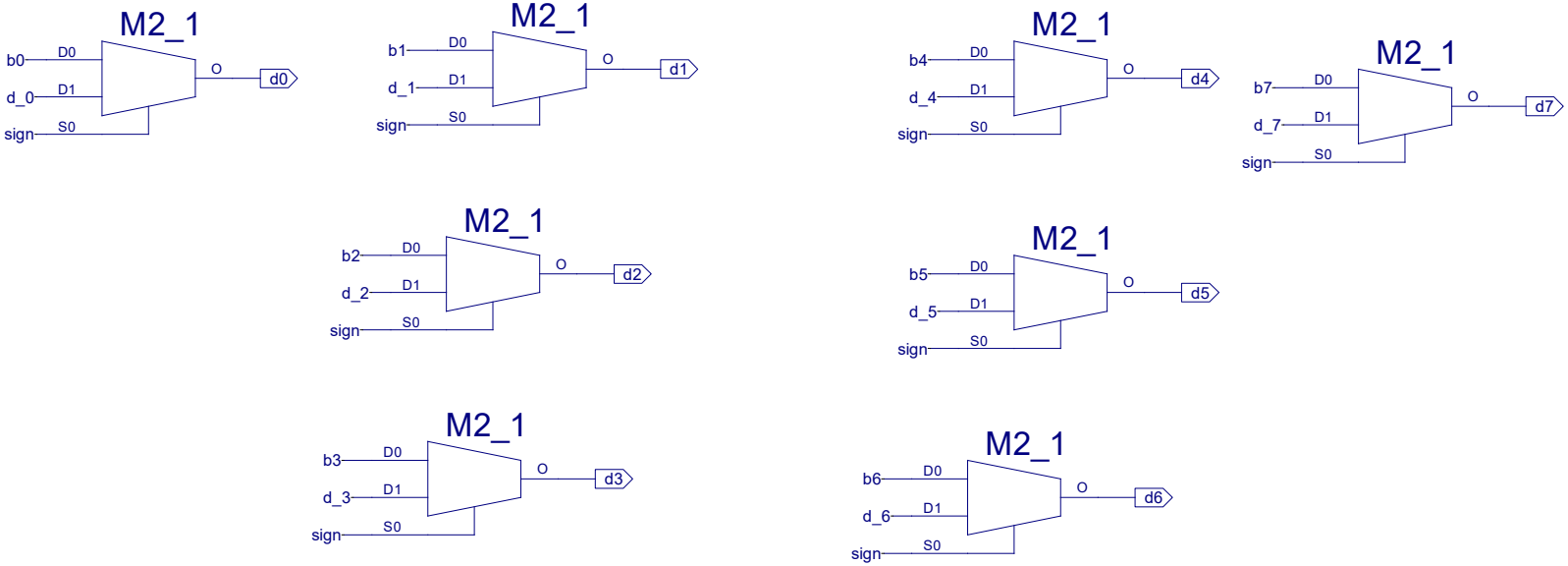
Sign Changer (2's Complement Calculator)



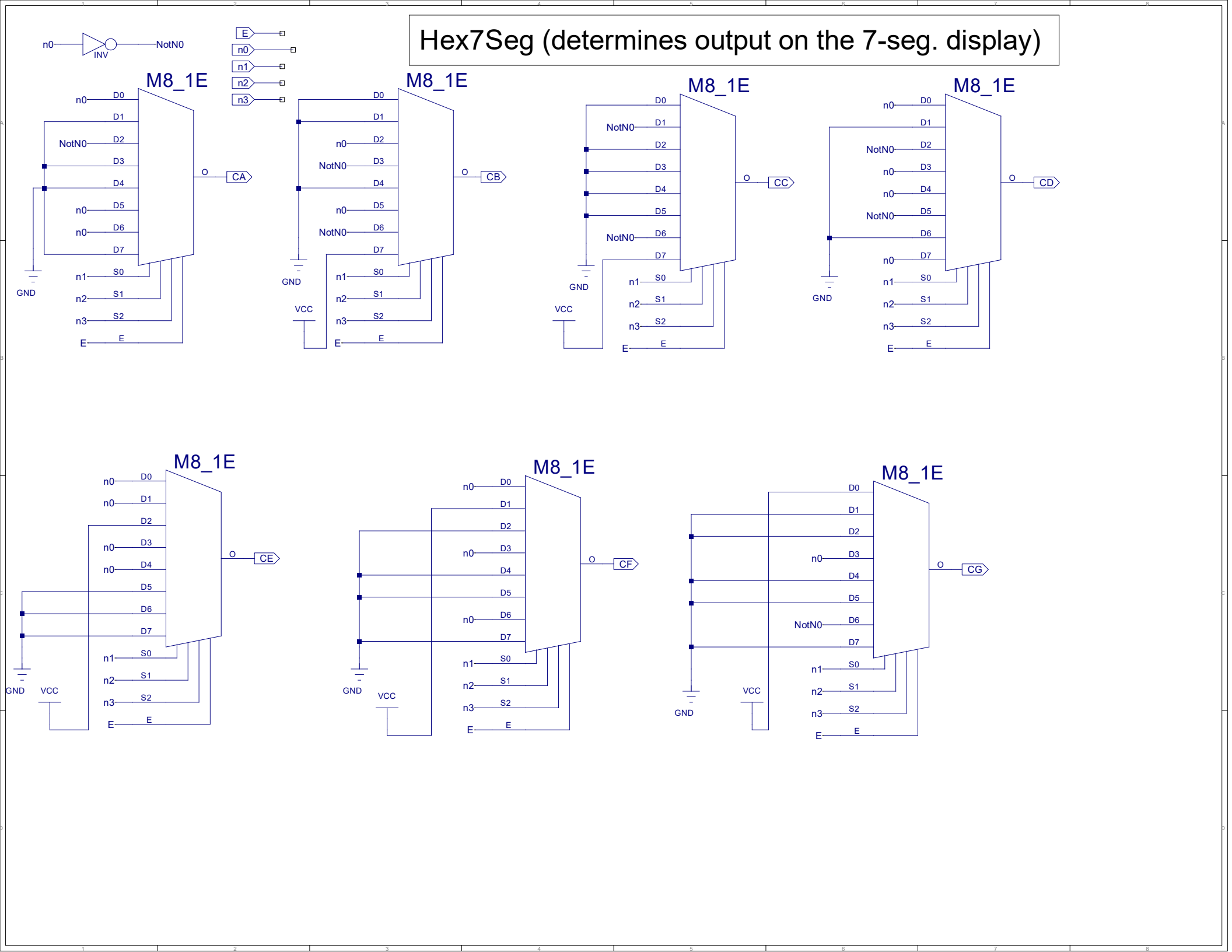




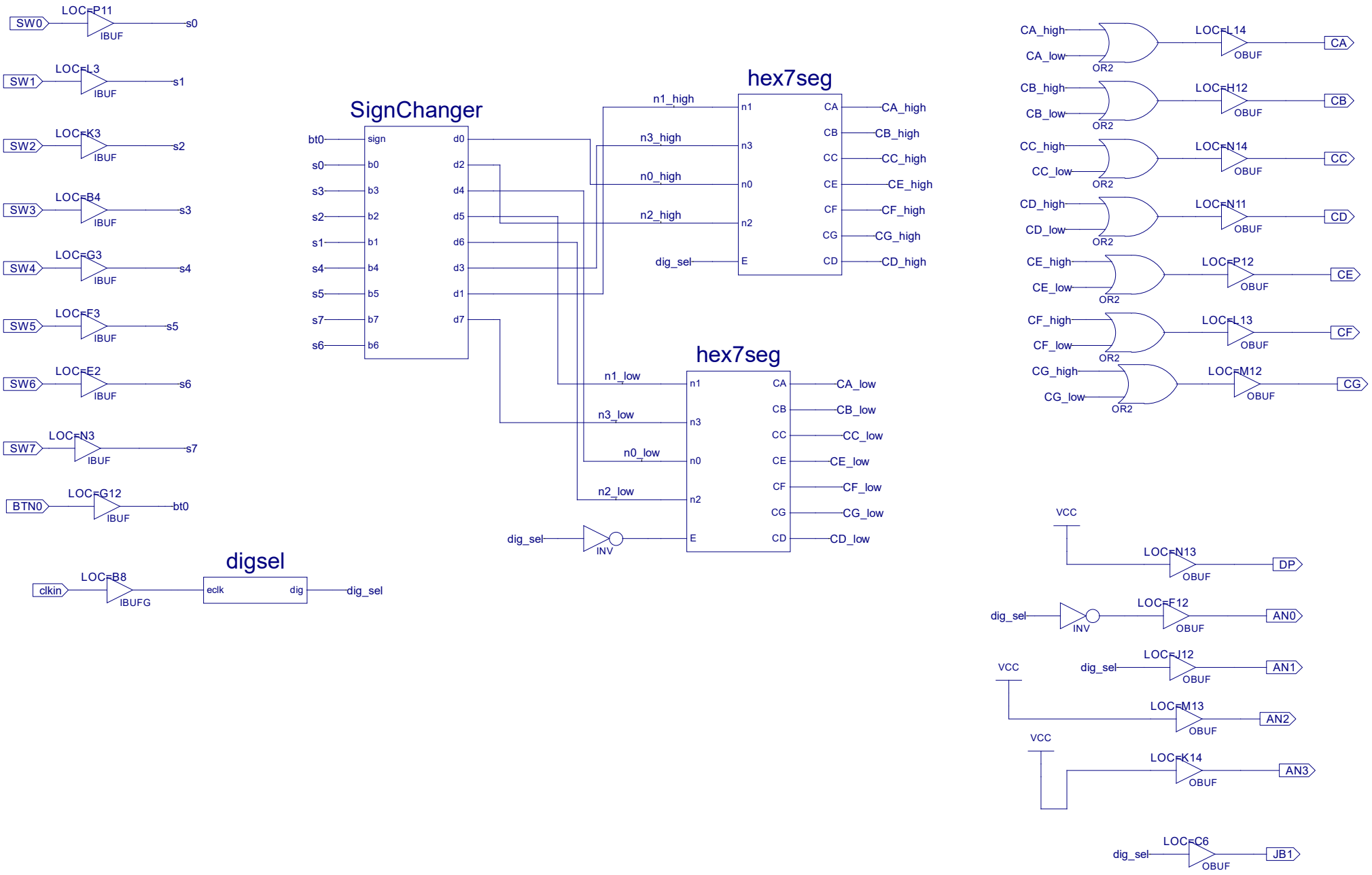




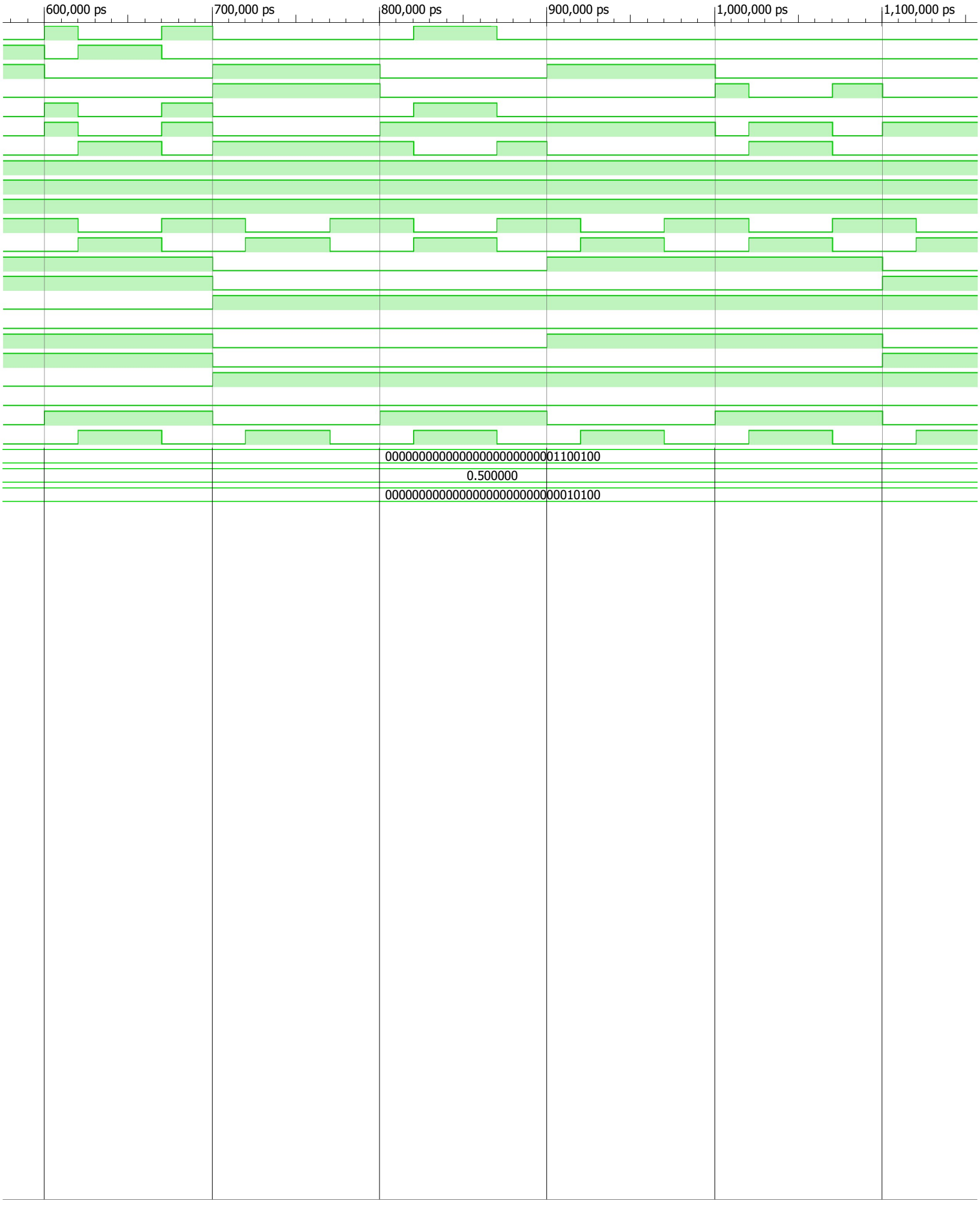
Hex7Seg (determines output on the 7-seg. display)

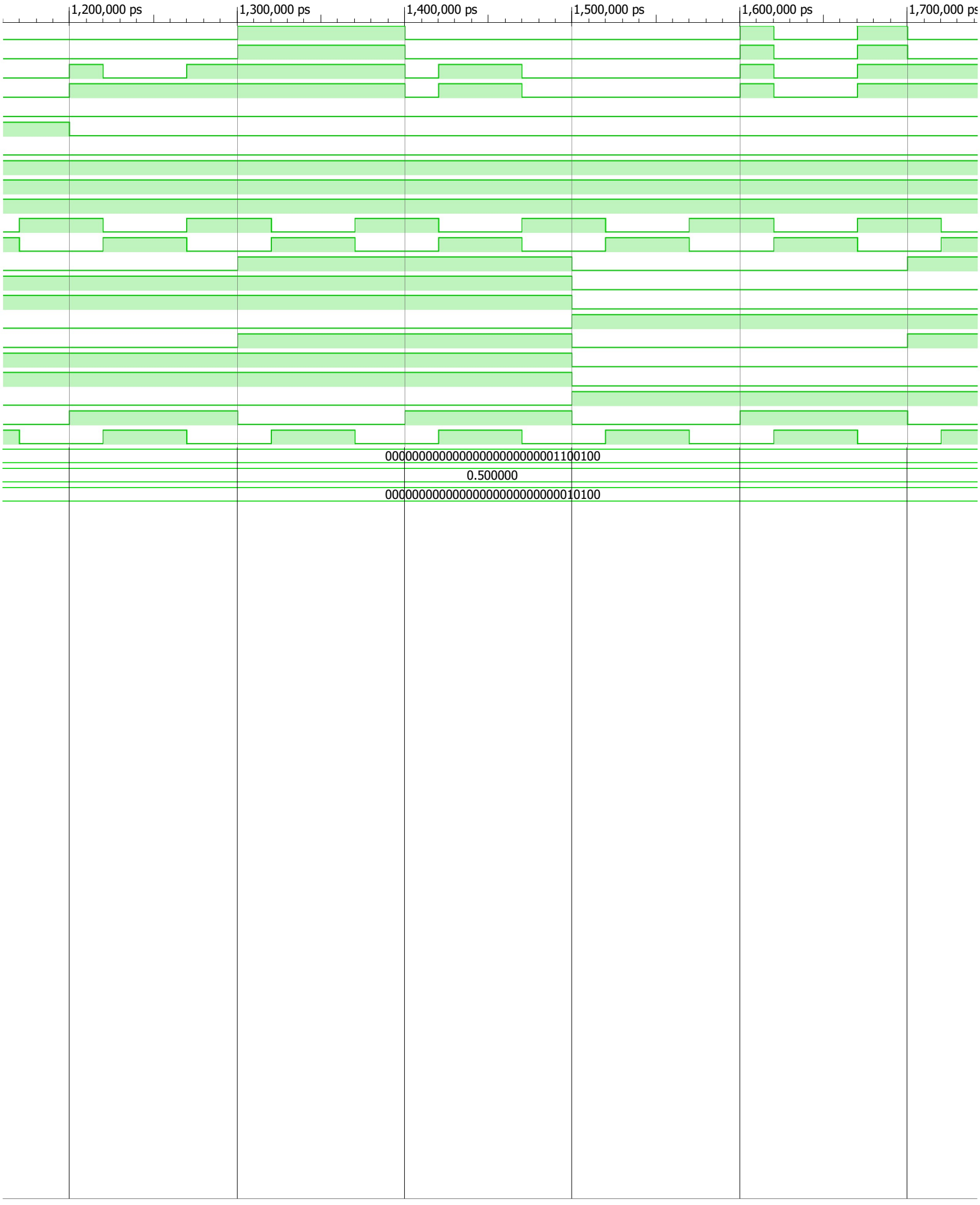


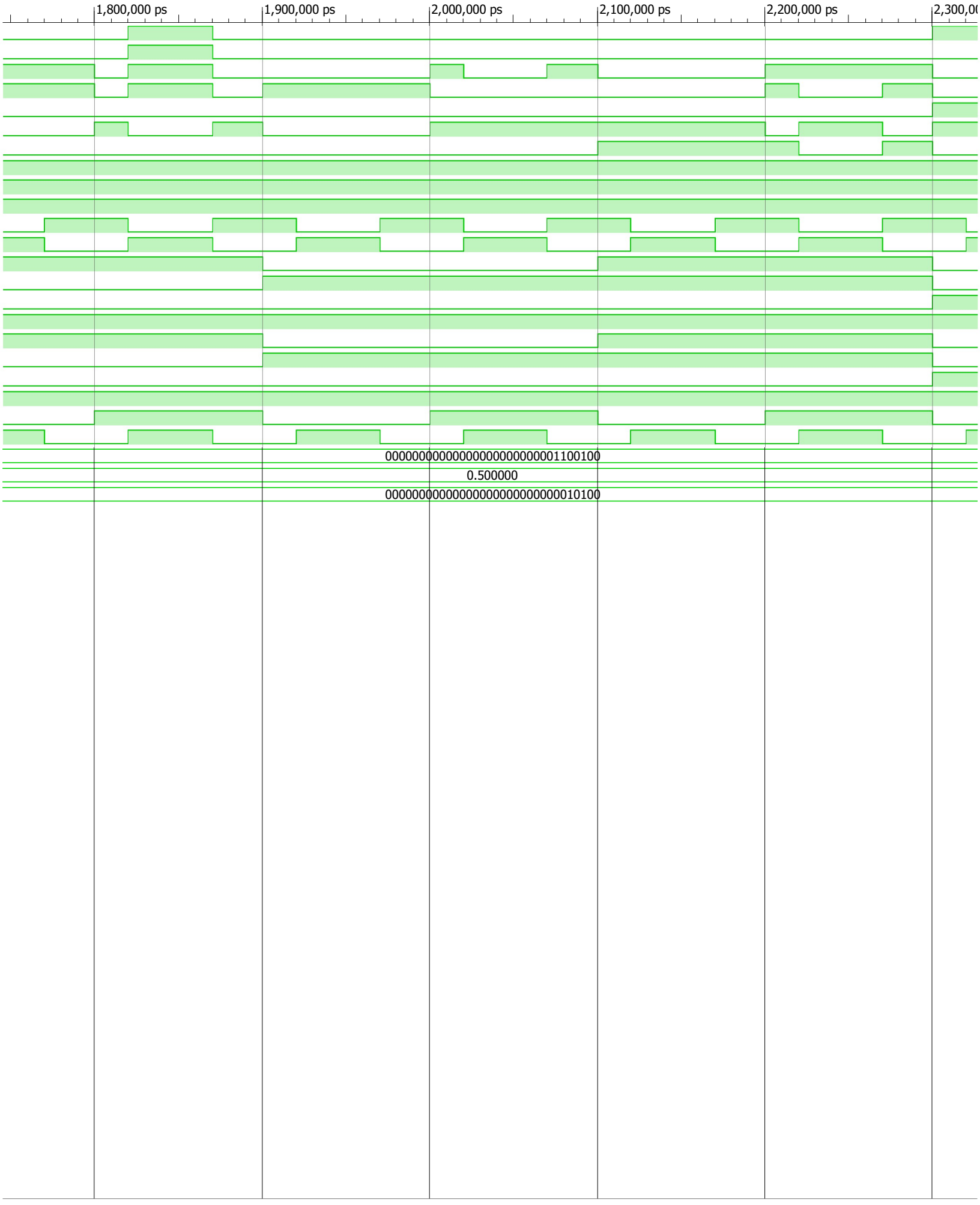
Top Level Schematic configures I/O and combines lower level schematics

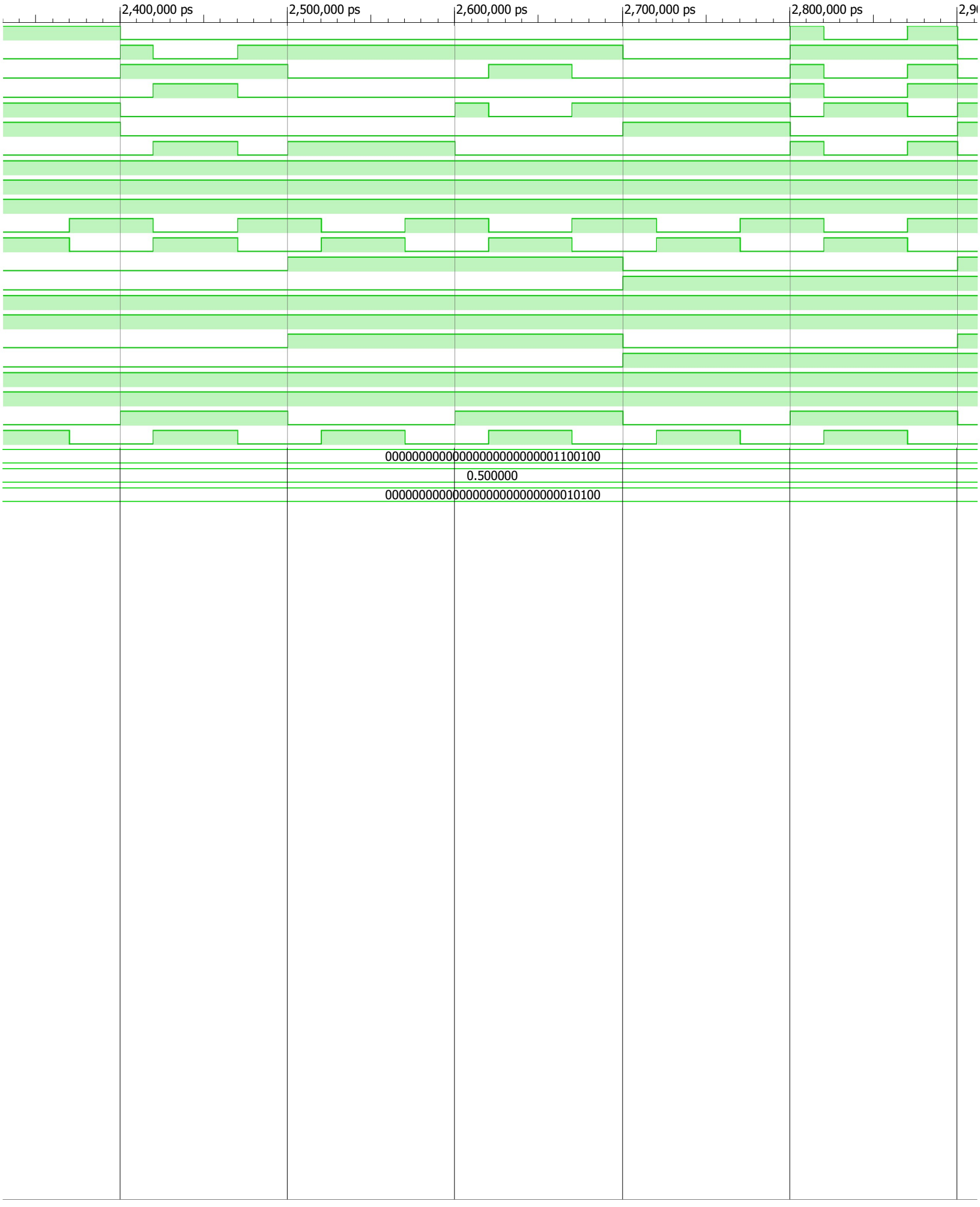


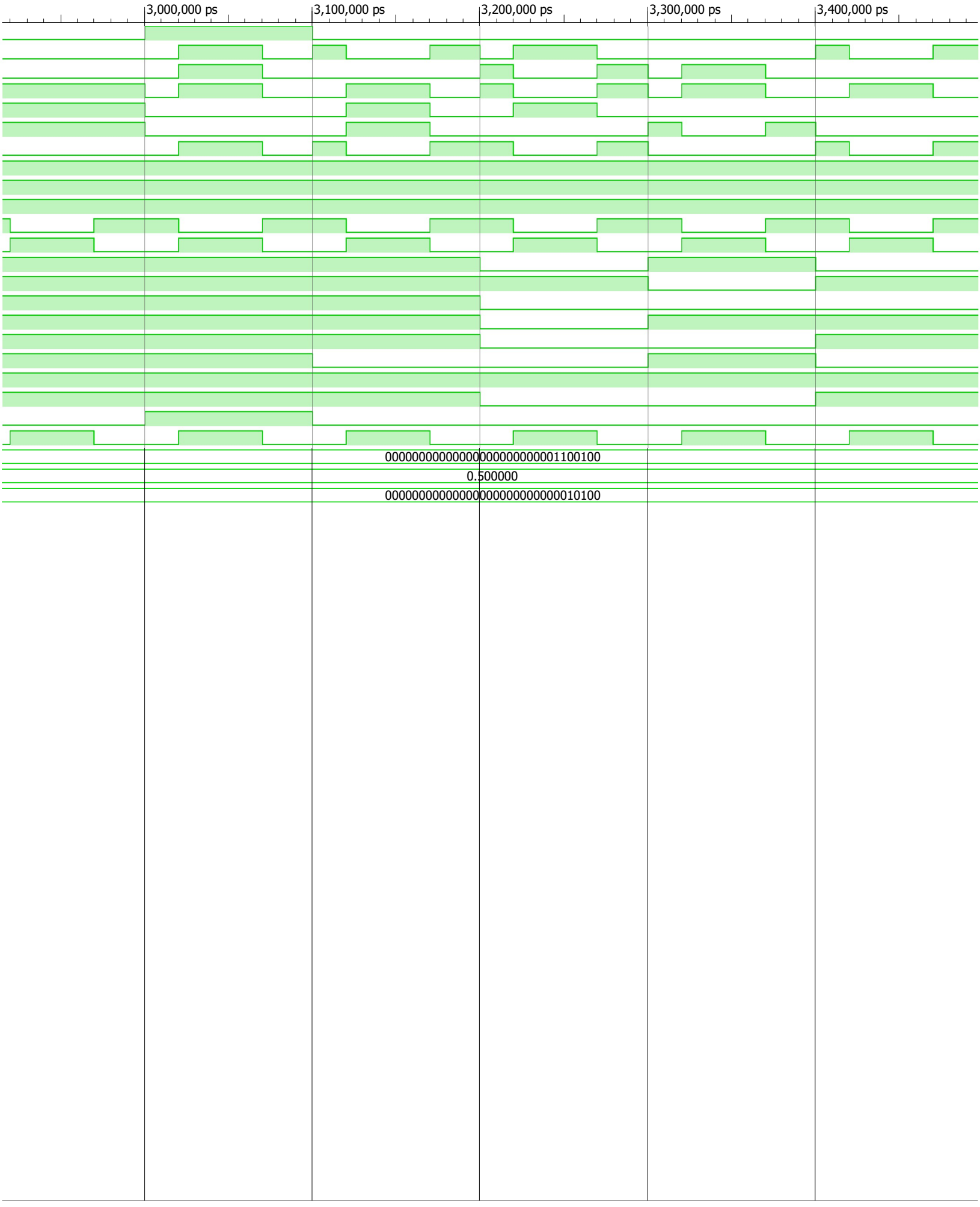

100,000 ps
200,000 ps
300,000 ps
400,000 ps
500,000 ps
CG
CF
CE
CD
CC
CB
CA
DP
AN3
AN2
AN0
AN1
SW0
SW1
SW2
SW3
SW4
SW5
SW6
SW7
BTN0
clkin
PERIOD[31:0]
00000000000000000000000001100100
DUTY_CYCLE
0.500000
OFFSET[31:0]
00000000000000000000000000010100


600,000 ps
700,000 ps
800,000 ps
900,000 ps
1,000,000 ps
1,100,000 ps
00000000000000000000000001100100
0.500000
00000000000000000000000000010100


1,200,000 ps
1,300,000 ps
1,400,000 ps
1,500,000 ps
1,600,000 ps
1,700,000 ps
00000000000000000000000001100100
0.500000
00000000000000000000000000010100


1,800,000 ps
1,900,000 ps
2,000,000 ps
2,100,000 ps
2,200,000 ps
2,300,00
00000000000000000000000001100100
0.500000
00000000000000000000000000010100


2,400,000 ps
2,500,000 ps
2,600,000 ps
2,700,000 ps
2,800,000 ps
00000000000000000000000001100100
0.500000
00000000000000000000000000010100

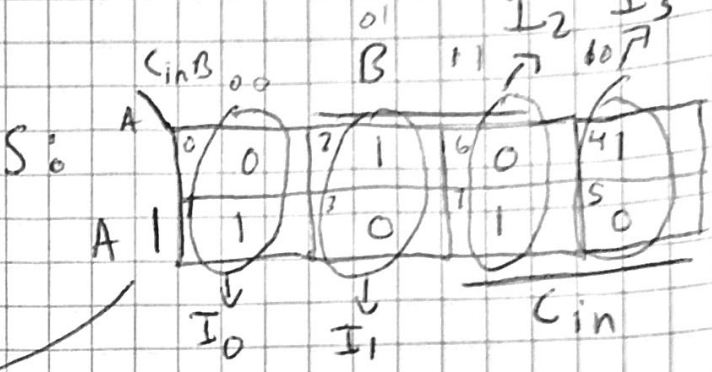

3,000,000 ps
3,100,000 ps
3,200,000 ps
3,300,000 ps
3,400,000 ps
00000000000000000000000001100100
0.500000
00000000000000000000000000010100

Lab 3

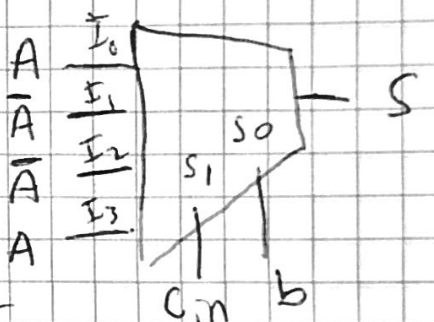
Adder Truth Table

C_{in}	B	A	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

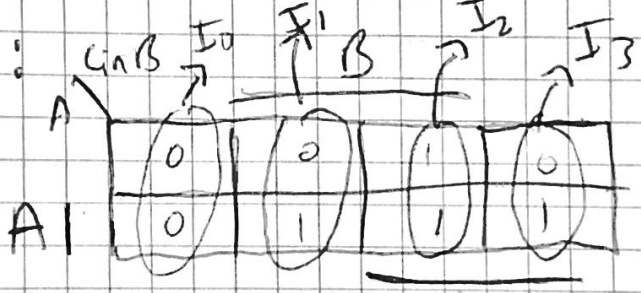
S:



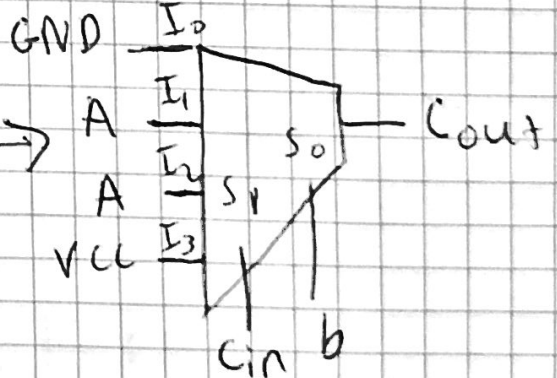
C_{in}	B	S
0	0	$I_0 = A$
0	1	$I_1 = \bar{A}$
1	0	$I_2 = \bar{A}$
1	1	$I_3 = A$



Cout:



C_{in}	B	C_{out}
0	0	$I_0 = 0$
0	1	$I_1 = A$
1	0	$I_2 = A$
1	1	$I_3 = 1$

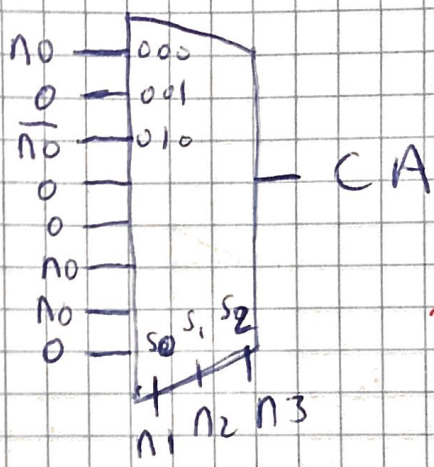


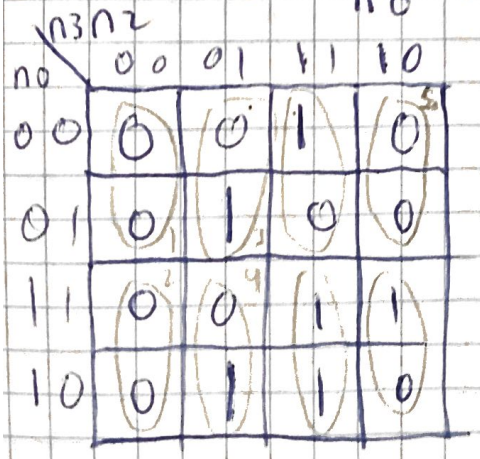
Vcc = 1
GND = 0

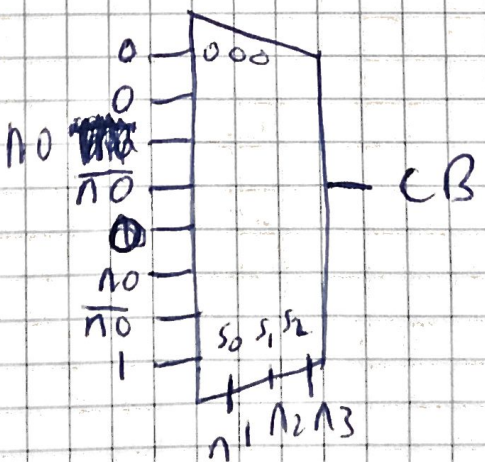
7-Segment Display

0 = on
1 = off

	n_3	n_2	n_1	n_0	CA	CB	CC	CD	CE	CF	CG	DP
0	0	0	0	0	0	0	0	0	0	0	1	1
1	0	0	0	1	1	0	0	1	1	1	1	1
2	0	0	1	0	0	0	1	0	0	1	0	1
3	0	0	1	1	0	0	0	0	1	1	0	1
4	0	1	0	0	1	0	0	1	1	0	0	1
5	0	1	0	1	0	1	0	0	1	0	0	1
6	0	1	1	0	0	1	0	0	0	0	0	1
7	0	1	1	1	0	0	0	1	1	1	1	1
8	1	0	0	0	0	0	0	0	0	0	0	1
9	1	0	0	1	0	0	0	1	1	0	0	1
A	1	0	1	0	0	0	1	0	1	0	0	1
B	1	0	1	1	0	1	1	0	0	0	0	1
C	1	1	0	0	0	1	1	0	0	0	1	1
D	1	1	0	1	1	0	0	0	0	1	0	1
E	1	1	1	0	0	1	1	0	0	0	0	1
F	1	1	1	1	0	1	1	1	0	0	0	1

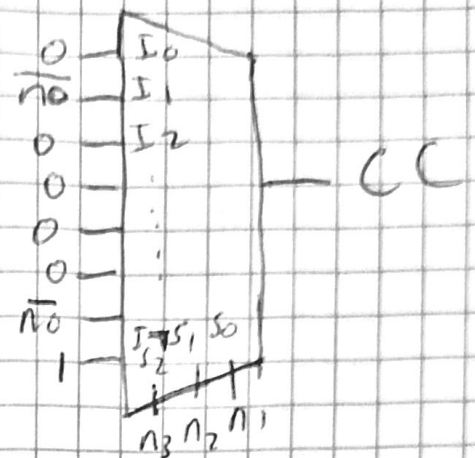




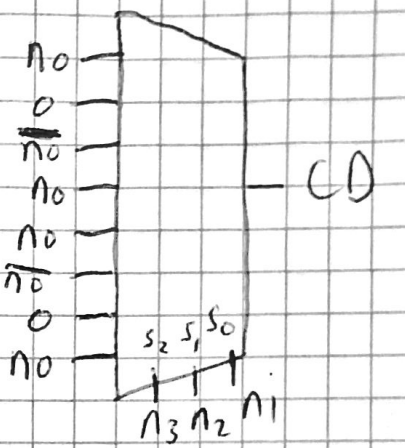


Tu 4/12
5358
Jyer

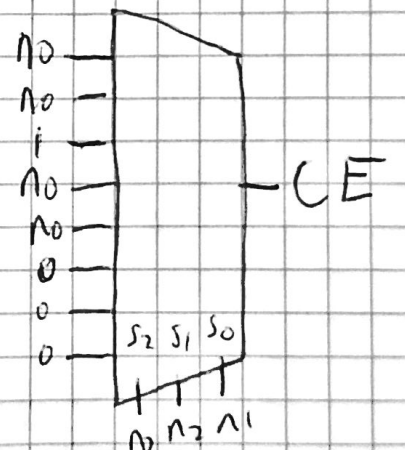
A_1A_0 \ A_3A_2	00	01	11	10
00	0	0	1	0
01	0	0	0	0
11	0	0	1	0
10	1	0	1	0



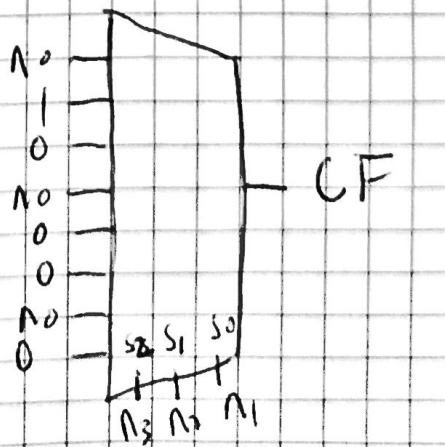
A_1A_0 \ A_3A_2	00	01	11	10
00	0	1	0	0
01	1	0	0	1
11	0	1	1	0
10	0	0	0	1



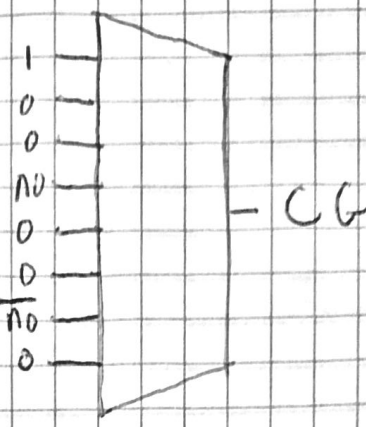
A_1A_0 \ A_3A_2	00	01	11	10
00	0	1	0	0
01	1	1	0	1
11	1	1	0	0
10	0	0	0	0



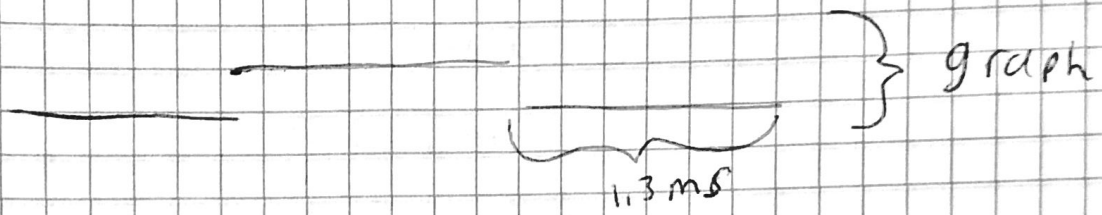
A_1A_0 \ A_3A_2	00	01	11	10
00	0	0	0	0
01	1	0	1	0
11	1	0	0	0
10	1	1	0	0



A1 A0 \ A3 A2	00	01	11	10
00	1	0	1	0
01	1	0	0	0
11	0	1	0	0
10	0	0	0	0



Oscilloscope



$P = 1.3 \times 10^{-3}$

$f = \frac{1}{1.3 \times 10^{-3}} = 769.2\ Hz$