|  |  |
| --- | --- |
| A picture containing electronics  Description generated with high confidence  Seven-Segment DISPLAY  Design and Implement a 7SD | Shady Boukhary, Brice Allard, Thomas Johnson  Logic Design 3023 |

7-Segment Display Information

The seven-segment display (7SD) consists of 7 LEDs arranged in a triangular fashion like the number 8. Every LED, or a segment, forms a part of a numerical digit when illuminated, whether it being a Decimal or a Hex digit. Moreover, the LED pins are labelled from a through g with each letter representing one LED or segment. Each of the decimal digits can be generated on the display by forward biasing – applying voltage across the diode in a way that the diode would allow the current to flow – the pins of the LEDs in a specific order. Consequently, some segments of the display will be dimmer than others, creating shapes. Every LED has two pins, a Cathode and an Anode. Therefore, this produces two types of 7SD displays called Common Cathode (CC) and Common Anode (CA), which is the more common one. The difference between the two is that the CC illuminate’s individual segments by applying high voltage – logical 1 - since initially all the cathode pins are connected to ground, while the CA illuminates them by applying low voltage – logical 0 – since all the anodes of the LEDs are initially connected to high voltage. The CA is more common since most circuits can usually sink voltage – ground – than they could provide. In other words, it’s easier and more efficient to drain voltage out by grounding pins already connected to Vin than it is to supply voltage to all these pins that are already grounded. To come up with a truth table for the seven-segment display, one needs to understand how to illuminate the segments to generate the digits first. To illuminate the display to generate the digit 0, 6 out of the 7 segments need to be illuminated. Particularly, the segments corresponding to the a, b, c, d, e, and f pins. The afore-mentioned pins would generate the numerical digit 0 by illuminating the segments that would resemble its shape. Applying the same logic to the rest of the numerical digits, one can deduce that in order to generate the numbers 1 through 9, the following display segments (LEDs) need to be illuminated respectively: b and c corresponding to 1; a, b, d, and e corresponding to 2; a, b, c, d, and g corresponding to 3; b, c, f, and g corresponding to 4; a, c, d, f, and g corresponding to 5; a, c, d, e, f, and g corresponding to 6; a, b, and c corresponding to 7; a, b, c, d, e, f, and g corresponding to 8 since all the LEDs need to be illuminated to represent 8; a, b, c, f, and g corresponding to 9. Moreover, to prevent damage to the LEDs, a resistor should be used. Typically, each LED segment can draw about 15 mA. Therefore, on a typical 5V circuit, a 200 Ohm resistor is needed by applying Ohm’s law on the circuit: where V is the voltage in Volts *(V)*, R is the resistance in Ohms *(Ω)*, and I is the current in Amperes *(A).* To get the resistance needed, we calculate . The truth table for a Common Anode 7-segment display derived using the above logic is presented in the next page in Figure 1.

Truth Table

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Decimal or Hex Digit | Individual LEDs (Segments) Illuminated | | | | | | |
| A | B | C | D | E | F | G |
| 0 *(0000)* | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 *(0001)* | 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| 2 *(0010)* | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 3 *(0011)* | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| 4 *(0100)* | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| 5 *(0101)* | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 6 *(0110)* | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 7 *(0111)* | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 8 *(1000)* | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 *(1001)* | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| A *(1010)* | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| B *(1011)* | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| C *(1100)* | 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| D *(1101)* | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| E *(1110)* | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| F *(1111)* | 0 | 1 | 1 | 1 | 0 | 0 | 0 |

*Fig. 1: Truth Table for a Common Anode (CA) 7-Segment Display with all possible values.*

VHDL Code for 7-Segment Display

library ieee;

use ieee.std\_logic\_1164.all;

entity sevensd is

port(x: in std\_logic\_vector (3 downto 0);

a: out std\_logic;

b: out std\_logic;

c: out std\_logic;

d: out std\_logic;

e: out std\_logic;

f: out std\_logic;

g: out std\_logic);

end;

architecture synth of sevensd is

begin

a <= (not x(3) and not x(2) and not x(1) and x(0)) or (not x(3) and x(2) and not x(1) and not x(0)) or (x(3) and x(2) and not x(1) and x(0)) or (x(3) and not x(2) and x(1) and x(0));

b <= (x(2) and x(1) and not x(0)) or (x(3) and x(1) and x(0)) or (not x(3) and x(2) and not x(1) and x(0)) or (x(3) and x(2) and not x(1) and not x(0));

c <= (not x(3) and not x(2) and x(1) and not x(0)) or (x(3) and x(2) and x(1)) or (x(3) and x(2) and not x(0));

d <= (not x(3) and not x(2) and not x(1) and x(0)) or (not x(3) and x(2) and not x(1) and not x(0)) or (x(3) and not x(2) and x(1) and not x(0)) or (x(2) and x(1) and x(0));

e <= (not x(3) and x(0)) or (not x(3) and x(2) and not x(1)) or (not x(2) and not x(1) and x(0));

f <= (not x(3) and not x(2) and x(0)) or (not x(3) and not x(2) and x(1)) or (not x(3) and x(1) and x(0)) or (x(3) and x(2) and not x(1) and x(0));

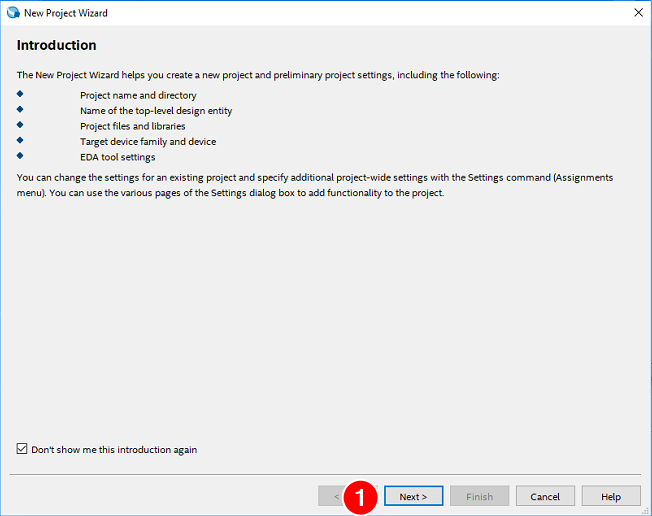
g <= (not x(3) and not x(2) and not x(1)) or (x(3) and x(2) and not x(1) and not x(0)) or (not x(3) and x(2) and x(1) and x(0));

end;

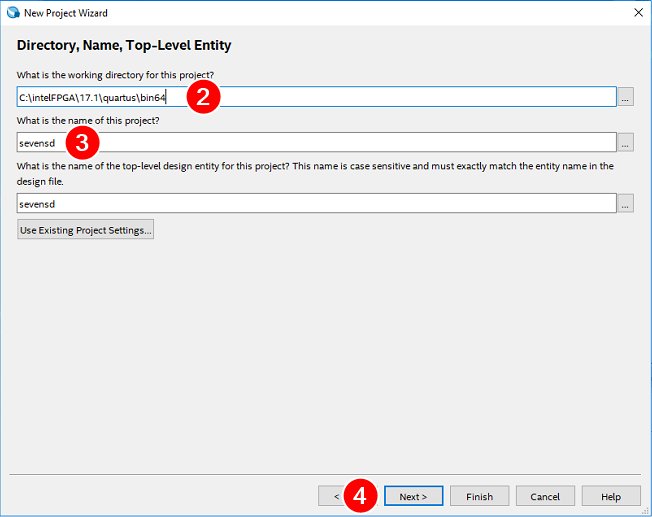
Quartus Prime / ModelSim – Altera Guide

We’ll be using Quartus Prime 17.1, but these steps can be used on any version of Quartus with minor differences.

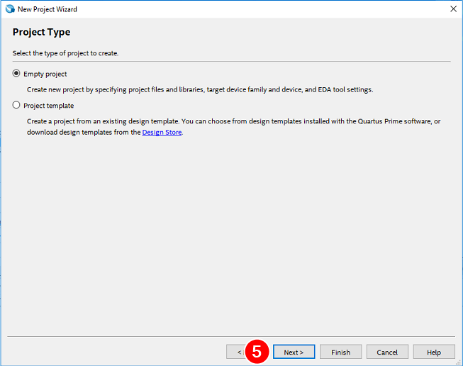
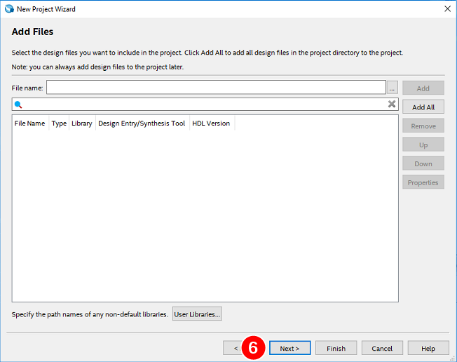
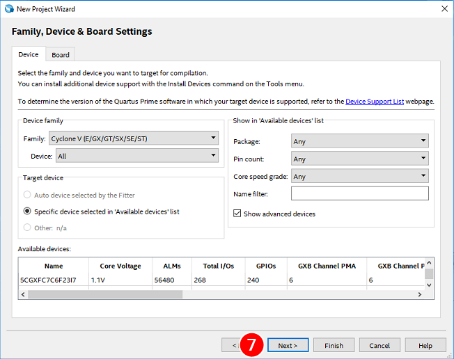
First, open up Quartus and go into the new project wizard.



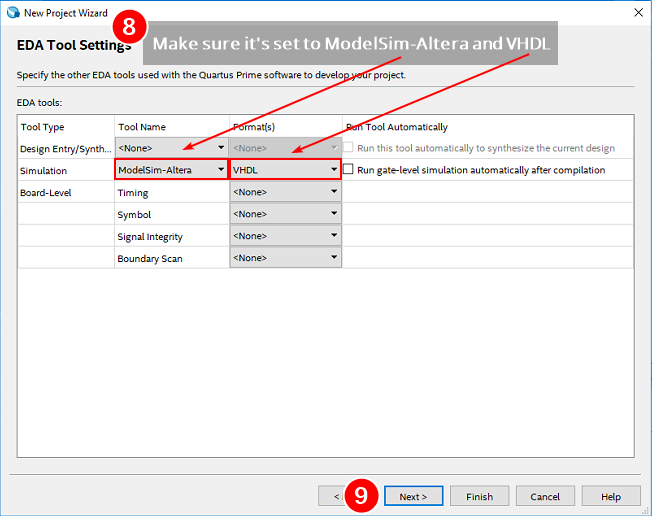
Enter whatever directory you please and choose a name for the project.



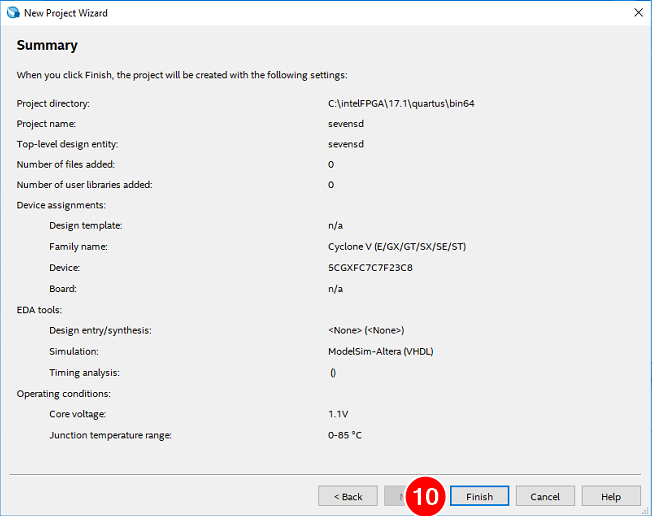
Continue through these prompts:



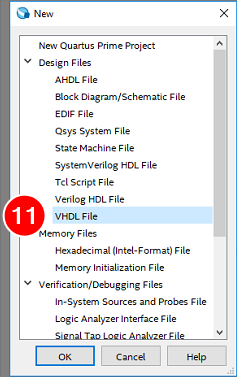
On the EDA Tool Settings, it is important to choose ModelSim-Altera and VHDL for the simulation to work.



Click finish to open a new project.



Now that you’ve created a new project, create a new VHDL file. File -> New



Copy and paste the following code into the file.

library ieee;

use ieee.std\_logic\_1164.all;

entity sevensd is

port(x: in std\_logic\_vector (3 downto 0);

a: out std\_logic;

b: out std\_logic;

c: out std\_logic;

d: out std\_logic;

e: out std\_logic;

f: out std\_logic;

g: out std\_logic);

end;

architecture synth of sevensd is

begin

a <= (not x(3) and not x(2) and not x(1) and x(0)) or (not x(3) and x(2) and not x(1) and not x(0)) or (x(3) and x(2) and not x(1) and x(0)) or (x(3) and not x(2) and x(1) and x(0));

b <= (x(2) and x(1) and not x(0)) or (x(3) and x(1) and x(0)) or (not x(3) and x(2) and not x(1) and x(0)) or (x(3) and x(2) and not x(1) and not x(0));

c <= (not x(3) and not x(2) and x(1) and not x(0)) or (x(3) and x(2) and x(1)) or (x(3) and x(2) and not x(0));

d <= (not x(3) and not x(2) and not x(1) and x(0)) or (not x(3) and x(2) and not x(1) and not x(0)) or (x(3) and not x(2) and x(1) and not x(0)) or (x(2) and x(1) and x(0));

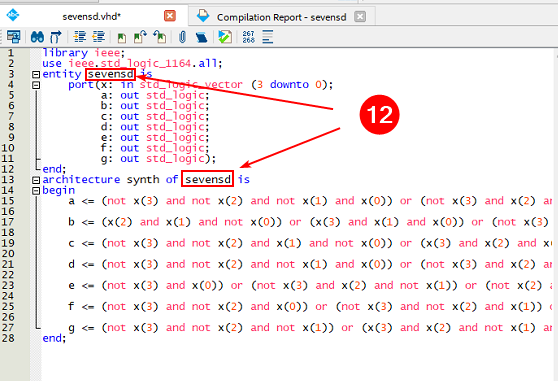
e <= (not x(3) and x(0)) or (not x(3) and x(2) and not x(1)) or (not x(2) and not x(1) and x(0));

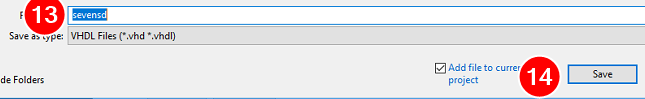
f <= (not x(3) and not x(2) and x(0)) or (not x(3) and not x(2) and x(1)) or (not x(3) and x(1) and x(0)) or (x(3) and x(2) and not x(1) and x(0));

g <= (not x(3) and not x(2) and not x(1)) or (x(3) and x(2) and not x(1) and not x(0)) or (not x(3) and x(2) and x(1) and x(0));

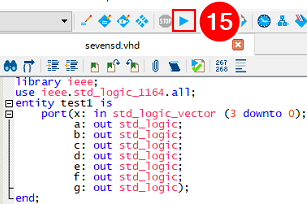
end;

Make sure the entity’s name matches its following architecture:



Now save it as the same name as your project.

Now that we have the VHDL file saved within the project, it’s time to compile it. Click the small blue play button.

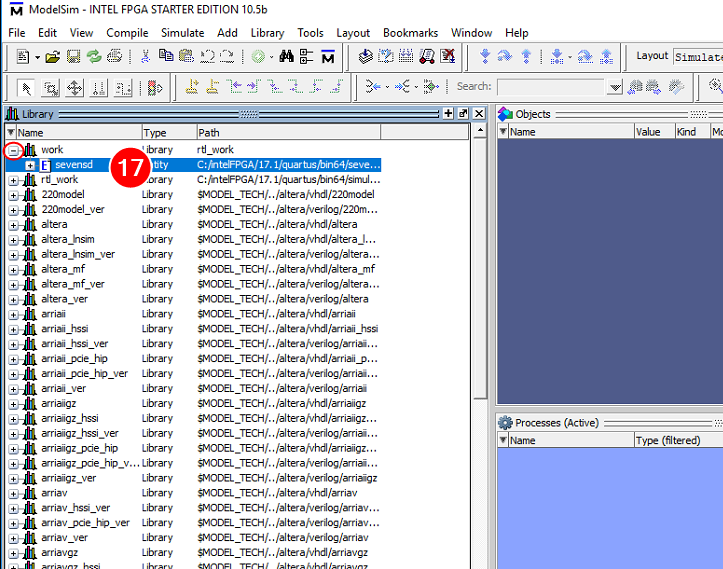


Quartus will compile the VHDL code for ModelSim to interpret and display in its simulation. Run the “RTL Simulation” as shown below.

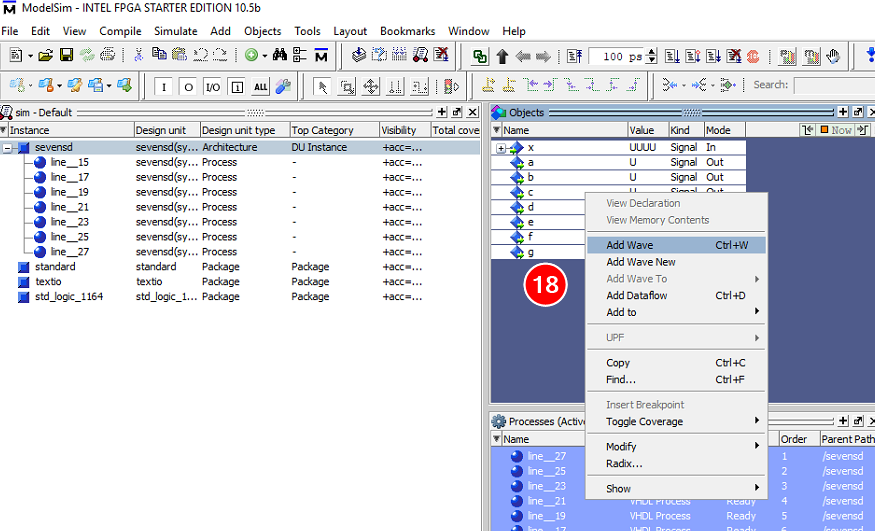
Tools -> Run Simulation Tool -> RTL Simulation



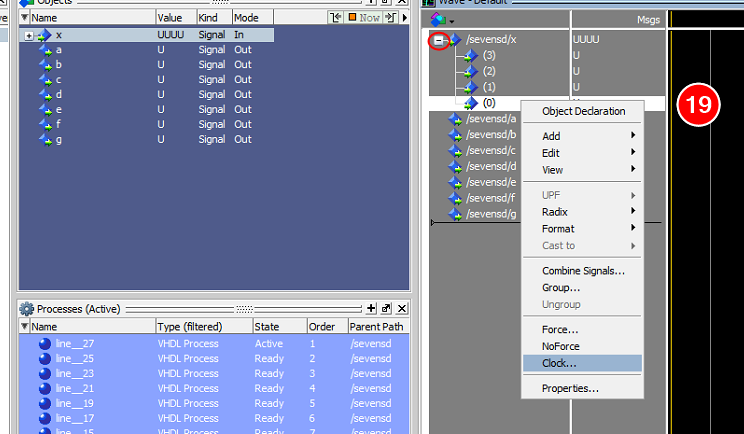
ModelSim-Altera will open the project. Expand the work row and double click the name of your project.



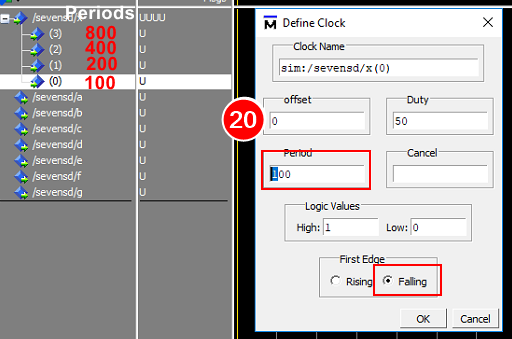
Select all inputs/outputs and add them to the waveform.



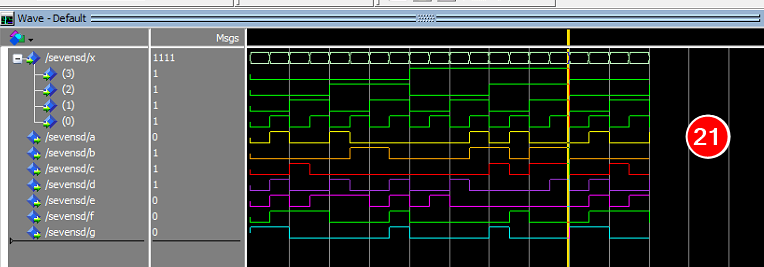
Now comes the tedious part. Expand the input vector. 0 is our LSB and 3 is our MSB. We need to adjust the clock rates of these three inputs to see every possible output.



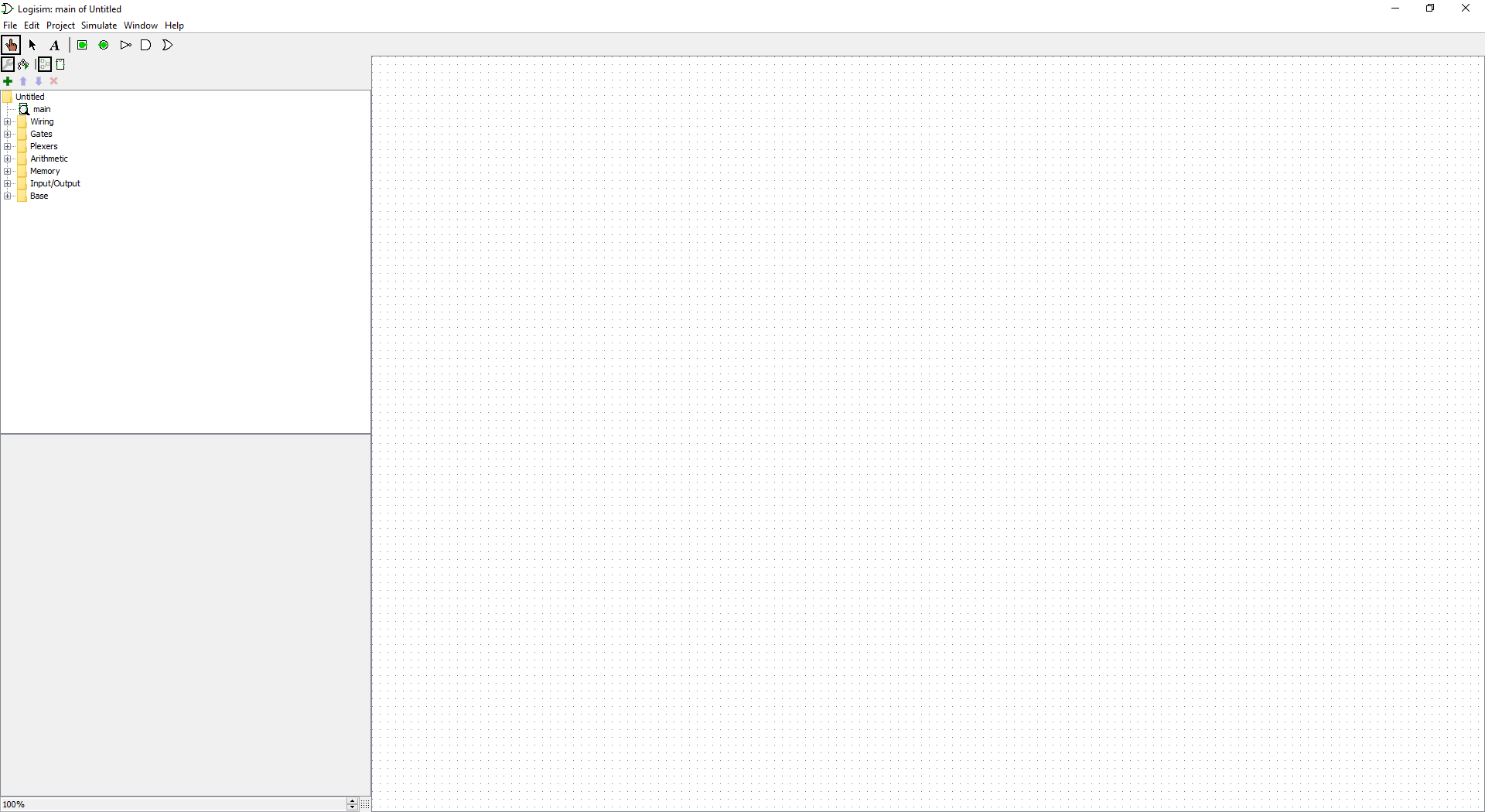
Each period configuration needs to be adjusted to the red text listed below **and** the first edge needs to be set to “Falling”



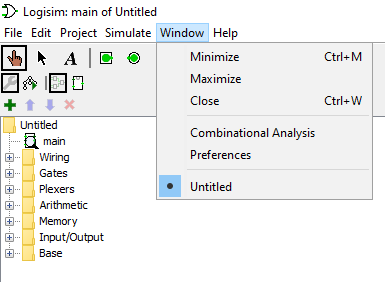
Once the input clocks have been set, you can change the colors or output names optionally. Pressing F9 will result in the waveform moving ahead 0.1ns. You will see every output from our VHDL code using this waveform every 0.8ns.



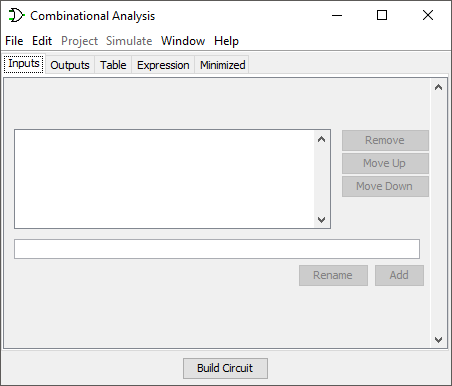
Logisim Guide

Let’s begin by opening Logisim and starting a new Project. When you open Logisim, it should look like this … 

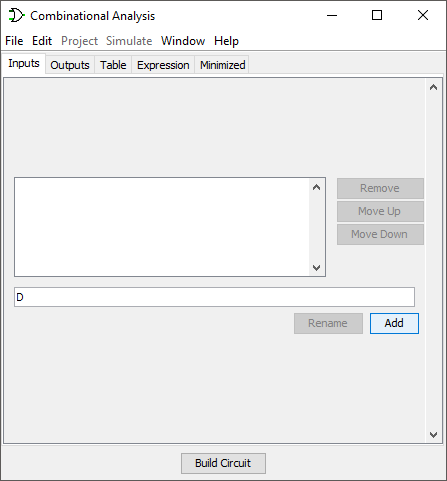
If not, Press CTRL+N to open a new project. Once a new project has been opened we begin by inputting the Combinational Analysis located in the Window drop down from the toolbar menu.



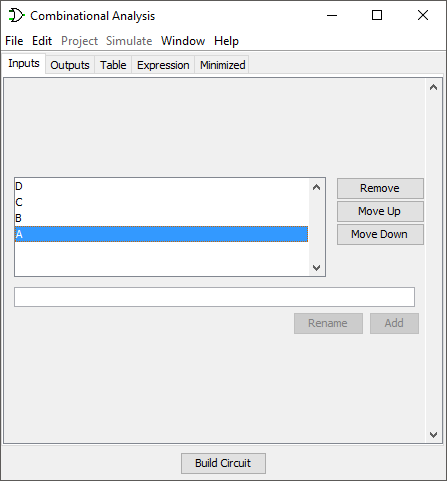
Which results in the following window being opened.



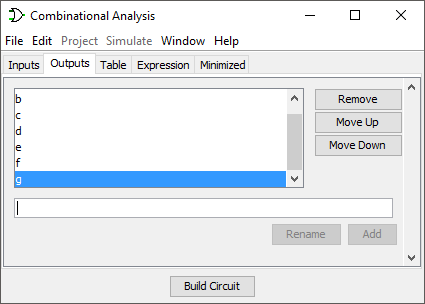
Now, using the Truth Table found previously, we notice that there are 4 inputs (D, C, B, A) that need to be entered into the analysis which will resemble the binary bit for its corresponding location of input. It should resemble the following …



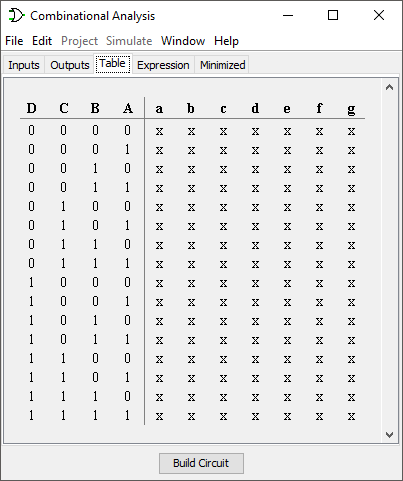
Ensure you are clicking the Add button after each letter to add it to the list of inputs. Please ensure you enter them in reverse order beginning with D and ending with A. I prefer to use capital letters to assure no confusion with the outputs in the next step. Resulting window should be as follows when completed correctly…



After the input we must enter how many outputs there will be from the previous Truth Table. This will simply be outputs a-g, also using the Add button to add them into the list. Starting with a and ending with g, the result should resemble …

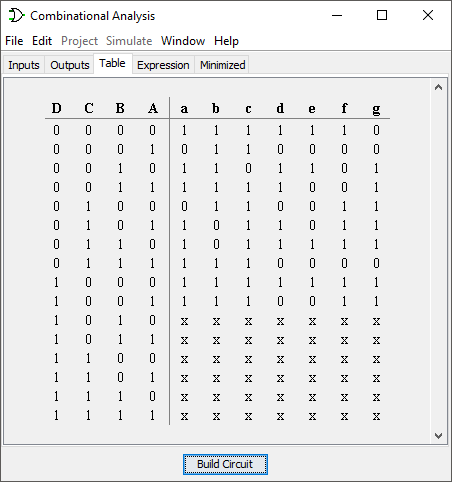


Now that our Inputs and Outputs have been chosen, we need to enter our truth table results from earlier into the Table tab. To do this you must first select the Table tab at the top of the screen which should result in a screen like this:

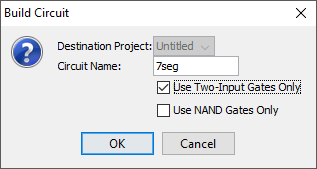


If your Table does not look like this, please revisit the previous steps to ensure accuracy.

Each ‘x’ can be clicked to change the results of the Truth Table. Let’s follow the truth table we provided earlier, making sure to flip each corresponding location with the opposite number (meaning 1’s should be 0’s and 0’s should be 1’s) because Logisim uses Common Cathode 7-segment displays by default and does not provide the pins necessary to choose. To explain, an actual 7-segment display would have 10 pins, 2 of which can be used to determine whether the display is being used as a Common Anode or a Common Cathode. However, in Logisim, only 8 pins are provided. Therefore, it is assumed that it is a Common Cathode (activated using 1’s). That is the reason the bits are flipped in the following table. The final product should be:

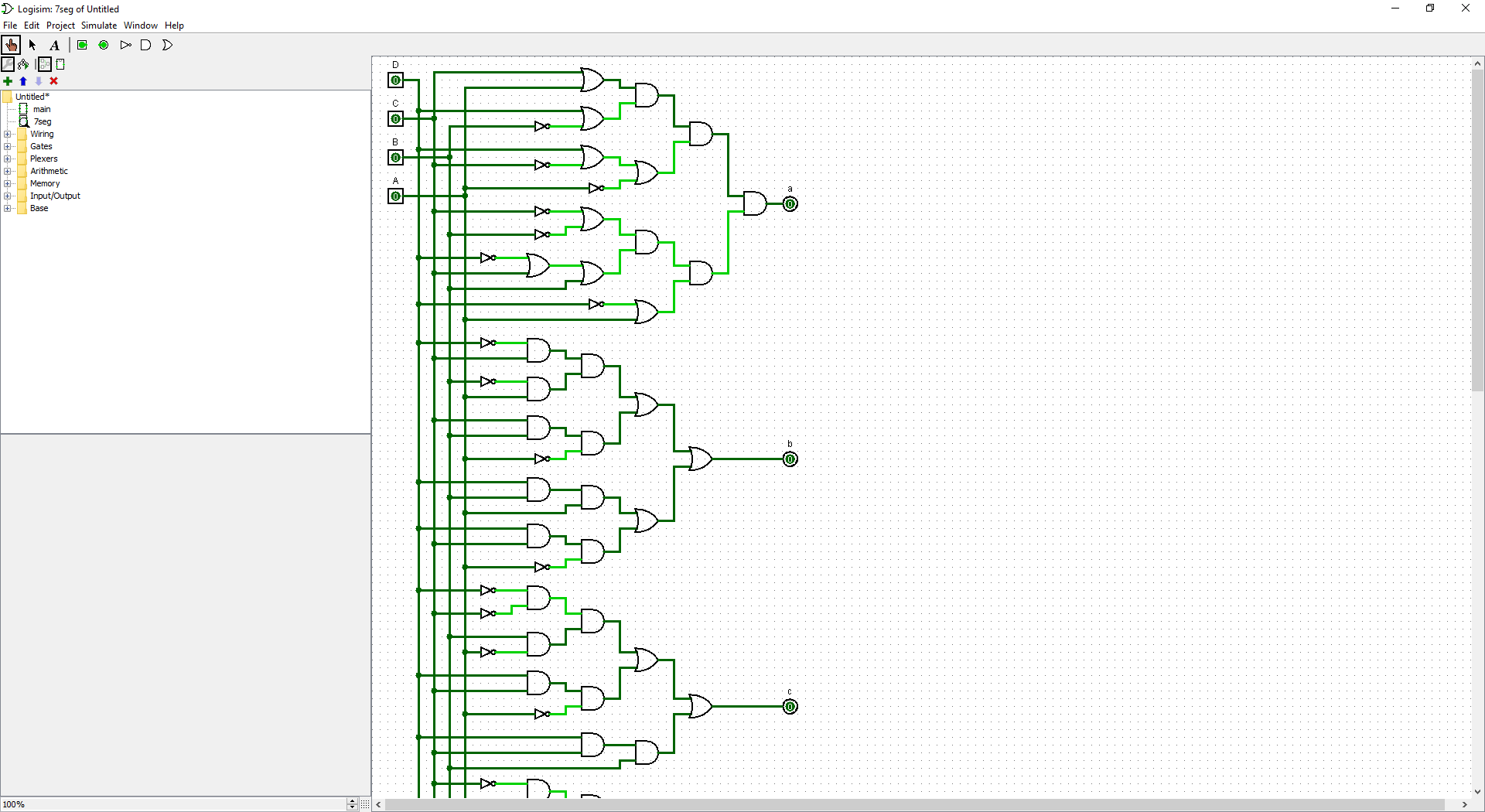


We can now click ‘Build Circuit’ and issue our circuit a name in the following popup:

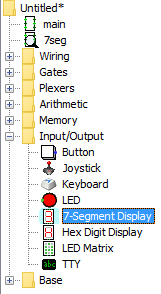


For this example, we will just enter “7seg” as our Circuit Name and ensure that ‘Use Two-Input Gates Only’ is checked, then press OK.

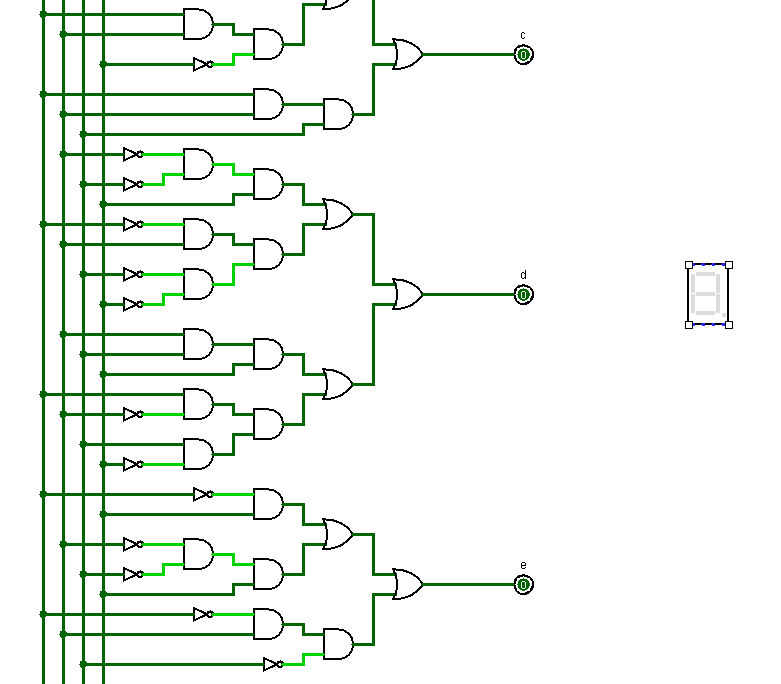
Your circuit is now generated on Logisim and should resemble:



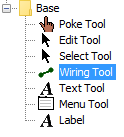
As you can see, our Inputs are located on the left and Outputs are located on the right. Now we must add a 7-segment display to our circuit. This can be done by located the appropriate addition in the left-hand side where multiple folders are located. You will find the 7-segment display in the ‘Input/Output folder.



Click on the 7-Segment Display and then move your mouse to the right side and place the display next to your outputs. In the example below, we place ours directly in the middle of our outputs to maintain a minimal design.



Now, for the next part we will be using the ‘Wiring Tool’ also located on the left-hand side like the previously discussed 7-Segment Display, however, this will be located in the ‘Base’ folder.

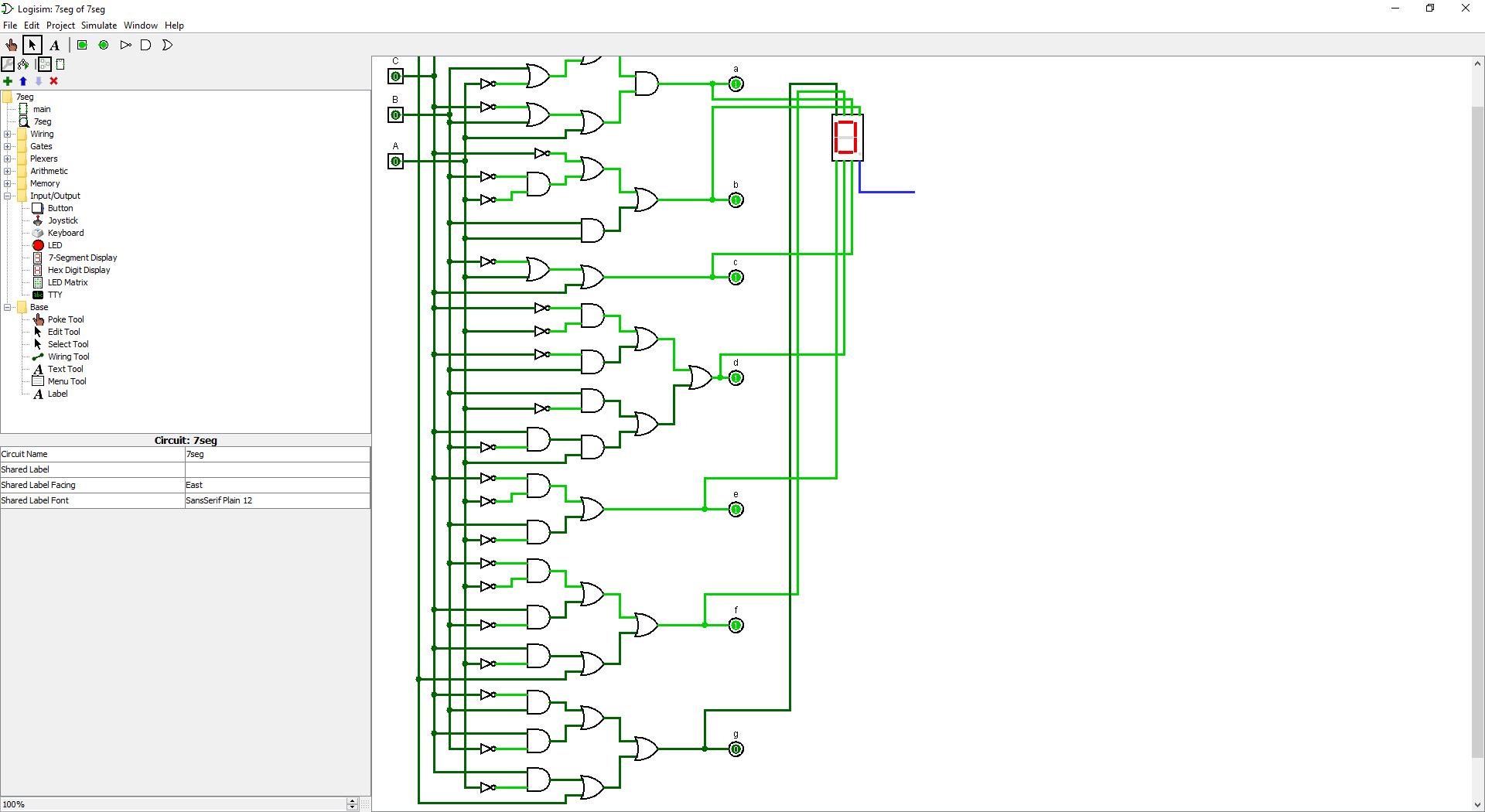


We must add a wire to each of the proper connections to the display for this to work properly. Follow the chart below precisely…

Each of the outputs above should be connected using the wiring tool to its associated position provided in the examples above, which should result in the following diagram being generated:



Your circuit is now complete. You can test each individual number by using the corresponding binary inputs for the number you would like displayed which are displayed below:

0 (0000), 1 (0001), 2 (0010), 3 (0011), 4 (0100), 5 (0101), 6 (0110), 7 (0111), 8 (1000), 9 (1001)

So for example, 6 will look like this:

 Resulting in being displayed.