

Complex logic function design



October 16, 2018

By Gary lin

# **Table of Contents**

[**Executive Summary** 2](#_Toc527553575)

[**Introduction 3**](#_Toc527553399)

[**Approach** 4](#_Toc527553765)

[**Electric Schematic** 8](#_Toc527553841)

[**LTSpice For Electric Schematic** 10](#_Toc527553856)

[**IRSIM For Electric Schematic** 13](#_Toc527553936)

[**Electric Layout For Give Function** 14](#_Toc527553944)

[**LTSpice Simulation For Electric Layout** 16](#_Toc527554093)

[**IRSIM Simulation For Electric Layout** 18](#_Toc527554100)

[**Conclusion** 20](#_Toc527554193)

[**References** 22](#_Toc527554205)

**Executive Summary**

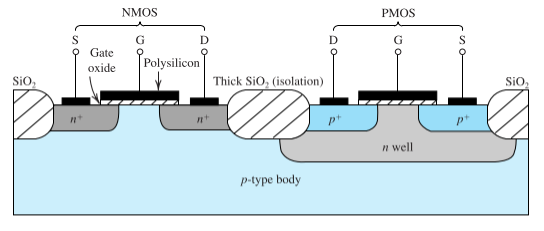
The goal of this project is to design a complex logic function using CAD tool, Electric and the function that needs to be designed is **((AB)’ + (B’C) + D’)’**. In order to build our function, we will be using CMOS transistors to design our network to output our function and we will simulate our network using simulation tools, LTSpice and IRSIM. Given a set of inputs, our simulations should output a graph that corresponds to the input at any given moment accurately.

The design of the complex logic function will consist of both a schematic and a layout for that schematic by utilizing Euler’s path and truth tables for confirmation of the correct output. The design of a complex logic function with CMOS transistors require a Pull Up Network (PUN) and a Pull Down Network (PDN) which work in duality in order to output the desired function. The given function requires a series of knowledge from the design of a NAND gate, inverter and complex functions.

The design would have three blocks of circuitry which are a block for each: the NAND gate, inverters and the complex function **G**. The NAND gate is used to inverter a function of two inputs and the inverters are used to invert our inputs which all go into the rest of our function **G**. Once that is done, two sets of inputs are used to test our design, one set to test every possible input and the other set which is given to be **(A, B, C, D) = (1,Pulse,1,0)**.

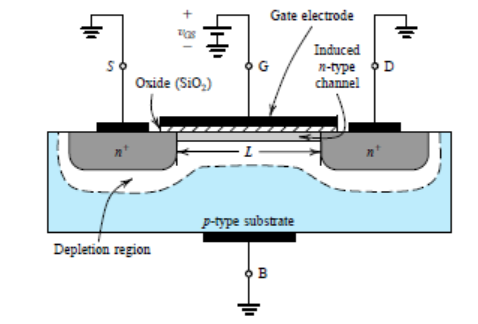
**Introduction**

Digital Integrated Circuits (ICs) are chips that are built using CMOS transistors to create certain functions that meet the requirements for specific outputs depending on the input to the IC. To break down what a CMOS transistor is, we look at the composition of a CMOS transistor as seen in *Figure 1* which consists of a combination of an NMOS transistor and a PMOS transistor. “While the NMOS transistor is implemented directly in the p-type substrate, the PMOS is fabricated in a special n-region known as the n-well. The two devices are isolated from each other by a thick region of oxide that functions as an insulator.” [1] To be able to make our layout, we must first understand how each of these transistors work.

****

*Figure 1: CMOS Transistor Breakdown*

An NMOS transistors can is one of the components of a CMOS transistor as seen in *Figure 2*. “It is fabricated on a p-type substrate, which is a single-crystal silicon wafer that provides physical support for the device. Two heavily doped n-type regions, in the figure as n+ source and n+ drain are created in the substrate. Then a thin layer of silicon dioxide (SiO2) of a thickness, usually denoted tox, which is an excellent electrical insulator, is grown on the surface, covering the surface and the drain. Metal contacts are then deposited on top of the oxide layer to form the gate electrode of the device. Metal contacts are also made to the source region, the drain region and the substrate, also known as the body. So there are four terminals, the gate (G), the source (S), the drain (D) and the body (B).” [2] An NMOS transistor is doped with Phosphorus to increase electron movement.



*Figure 2: NMOS Transistor*

As for a PMOS transistor, it is similar to an NMOS transistor but instead of an p-type substrate, an n-type substrate is used as the body and it is doped with Boron in order to create holes. Now that we know the design of both NMOS and PMOS transistors, we look back at the CMOS transistor. The CMOS transistor is created using a p-type substrate but an n-well is created in order to be able to integrate a PMOS transistor into it.

**Approach**

The given complex logic function is not as simple as it seems at first glance so the first step is to take a little time to think about the whole process and compare it to simpler complex logic functions that can be seen in textbooks or found online. The difference between the given function and functions seen in textbooks is the part **(A\*B)’** because the output of the entire network is negated rather than individual parts which we would require an inverter to do. But it can be easily seen that instead of using an inverter to negate the function **A\*B**, we can just use a NAND gate to create the needed function. Doing so will give us a function **F = (AB)**’ and we connect it the other part of the function to get **G = (F+(B’\*C)+D’)’** which would give us the function that we need.

Now that we have figured out how to design our function, we need to look at the theoretical output of the function. In order to do so, we need to create a truth table and we would need the output for every possible input. With our truth table seen in *Figure 4,* we can see that there’s only 2 possible sets of inputs that would give us an output of 1.

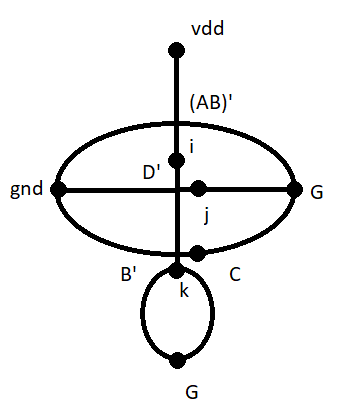
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **A** | **B** | **C** | **D** | **Out** |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |

*Figure 5: Truth Table for Complex Logic Function*

There’s a possibility that there may be an error while computing the output so to make sure that the truth table gives us the expected output, we can easily simplify the function to a simpler form. If we use a combination of DeMorgan’s Law, Distributive Law and Identity OR Law, we can simplify our function to **G = ABD** and from this function, we can see that there is only 1 set of inputs that gives us an output of 1 which is **A=1**, **B=1** and **D=1**. So why do we see two outputs in our logic function that gives us an output of 1? The reason behind that is simple, we need to account for the input **C**. Since the output of the function isn’t affect by **C**, that means we should get an output of 1 when we have the input sets **(A,B,C,D) = {(1,1,0,1), (1,1,1,1)}** which is exactly what we see in our truth table.

With the given set of inputs, **(A, B, C, D) = (1,Pulse,1,0)**, we can see that our output will result in 0 outputs according to our truth table and we can also see this from the simplified function, **G = ABD**. We can see that there’s only one case in which **G=1** and if we look at our input without **C**, we get that **G=0** because our inputs would be the set, **(A,B,D) = (1,1,0)**. Our input **D** is always 0 which results in a 0 no matter what the value for **A** and **B** we have. So the expected output of the set of inputs would result in 0.

With our complex logic function thought out, our schematic can easily be designed but our layout will still pose a challenge for us. To make our layout easier to bear, we use a technique called Euler’s path to help us decide the orientation of our layout. Euler’s paths must satisfy both our PUN and our PDN otherwise, our layout will be separated into many different sections which is why Euler’s path helps us simplify the our challenge of creating our layout. As we can see from *Figure 6*, each transistor is represented by an edge and each node is represented by a vertex and in order to decide which paths are possible, we need to obtain the paths that move along each edge only once otherwise, it would be an invalid path.

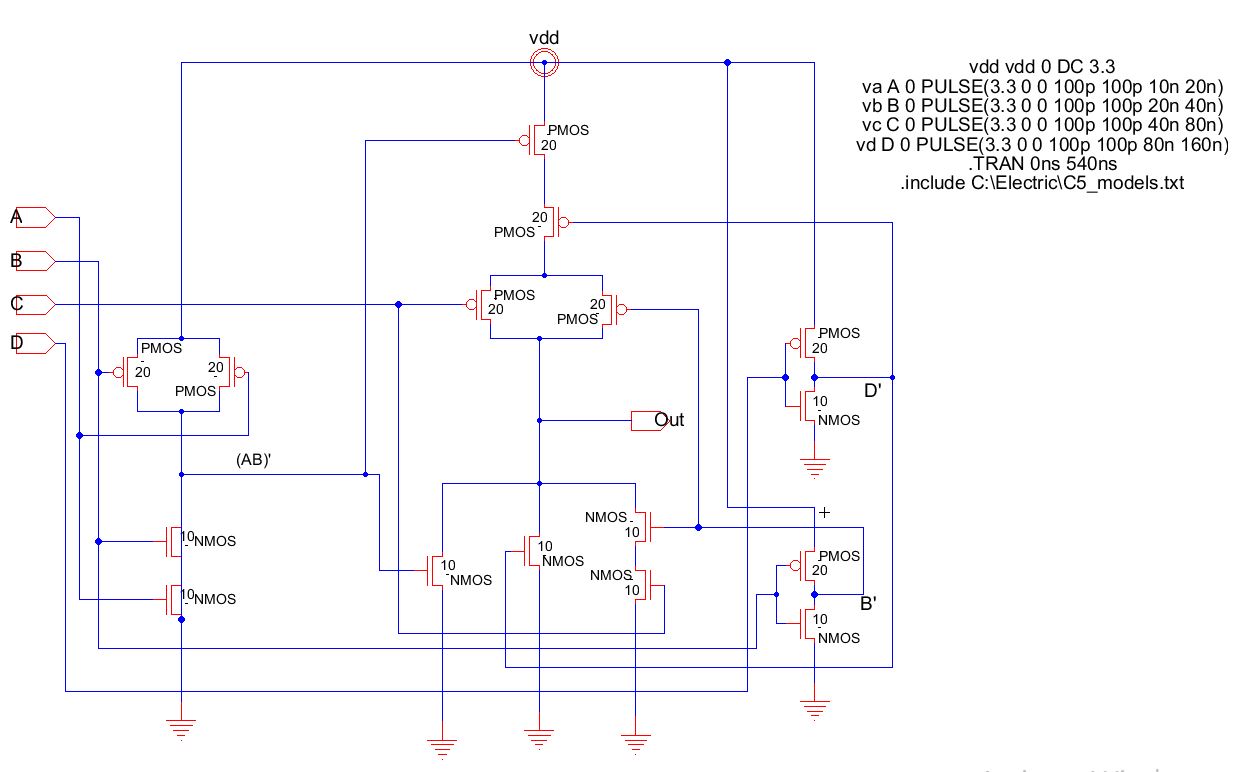


*Figure 6: Euler’s Path for Complex Logic Function*

In order to obtain our possible paths, we use trial and error and in the end we get that the following paths are possible: **(AB)’D’CB’**, **(AB)’D’B’C**, **B’CD’(AB)’**, **CB’D’(AB)’**

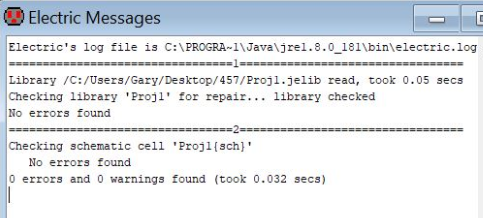
For the design of the layout, the Euler path, **(AB)’D’CB’** will be used and it would contain three parts which is for the NAND gate, two inverters for inverted outputs and the function **G**.

**Electric Schematic**

Now that we have the complex logic function design all figured out, we can move onto design our schematic. Our schematic will consist of the specified three parts, NAND gate, two inverters and function **G**. When we put it all into our schematic, we get the image seen in *Figure 7*. 

*Figure 7: Electric Layout of Complex Function*

We can see that the output of the NAND gate, **(AB)’**, goes into a transistor that connects with the rest of the function **G** and we have the two inverters on the right side of the schematic to inverter our inputs **B** and **D** before we can transfer it into our transistors. Once we have our schematic designed, we have to make sure that there aren’t any errors with the design using our Electric function of error checking. Our error check shows us that there aren’t any errors with the design as seen in *Figure 8*.

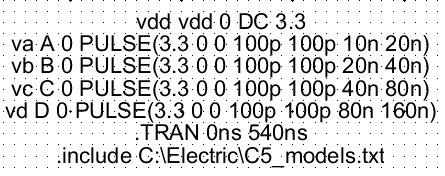


*Figure 8: Error Check for Schematic*

Now that we have our design, we move on to testing the design to see if we get our desired output.

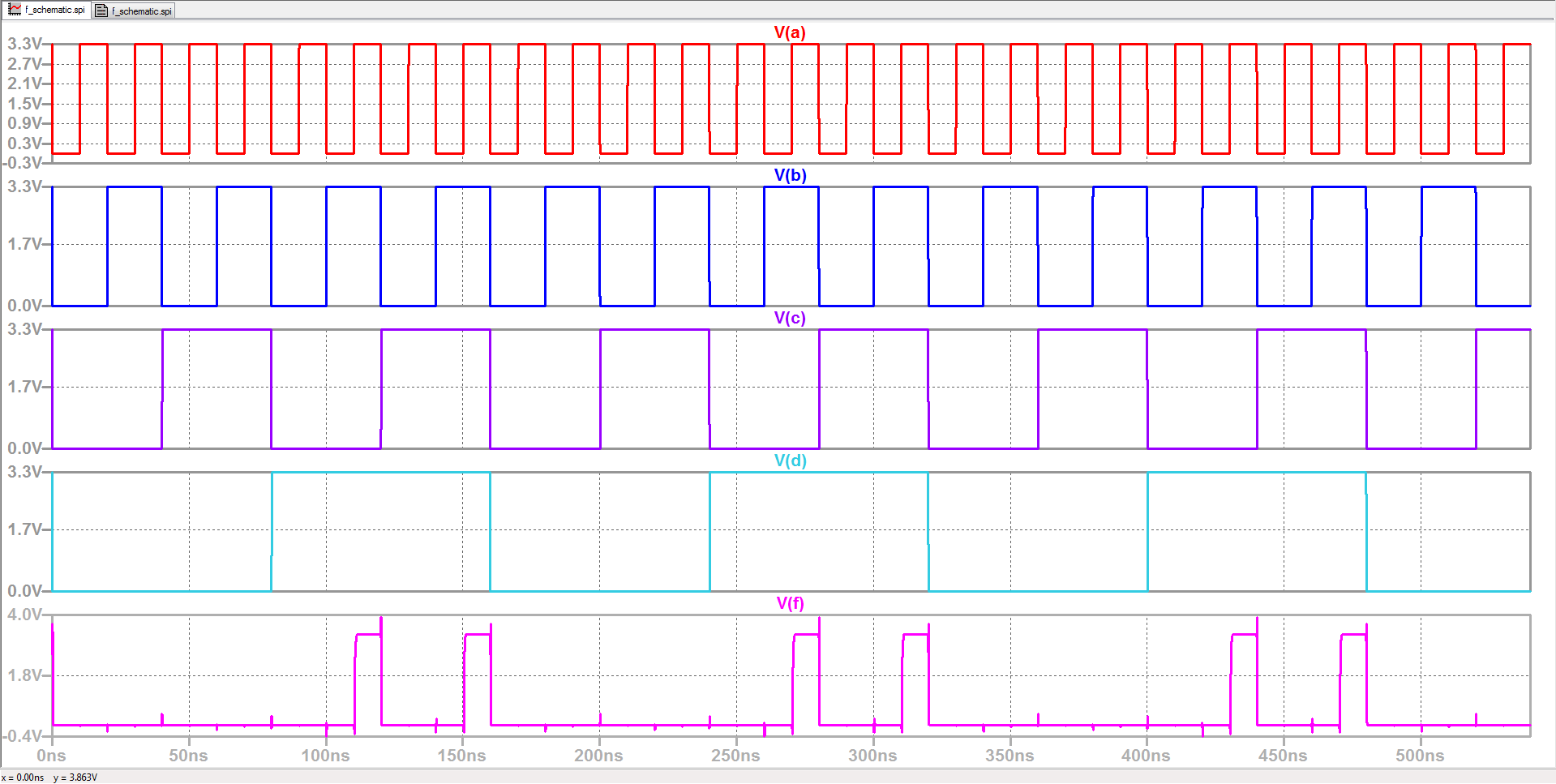
**LTSpice for Electric Schematic**

The output would be determined by two sets of inputs, a set of inputs that tests every possible case and a set of inputs that are given, **(A, B, C, D) = (1,Pulse,1,0)**. As discussed before, the second set of inputs would result in 0 while the first set of inputs would result in two spikes in the graph per period. We test the first set of inputs, where **A,B,C,D** are all pulse functions with different frequency as one would see in a four bit counter. We send in the Spice Code seen in *Figure 9*, and we should get an output that we see in our truth table.

****

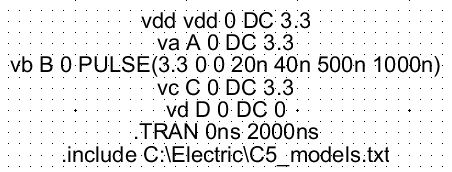
*Figure 9: Spice Code for All Possible Inputs*

With our Spice Code set up, we run our transient analysis, we get the output seen in *Figure 10*.

****

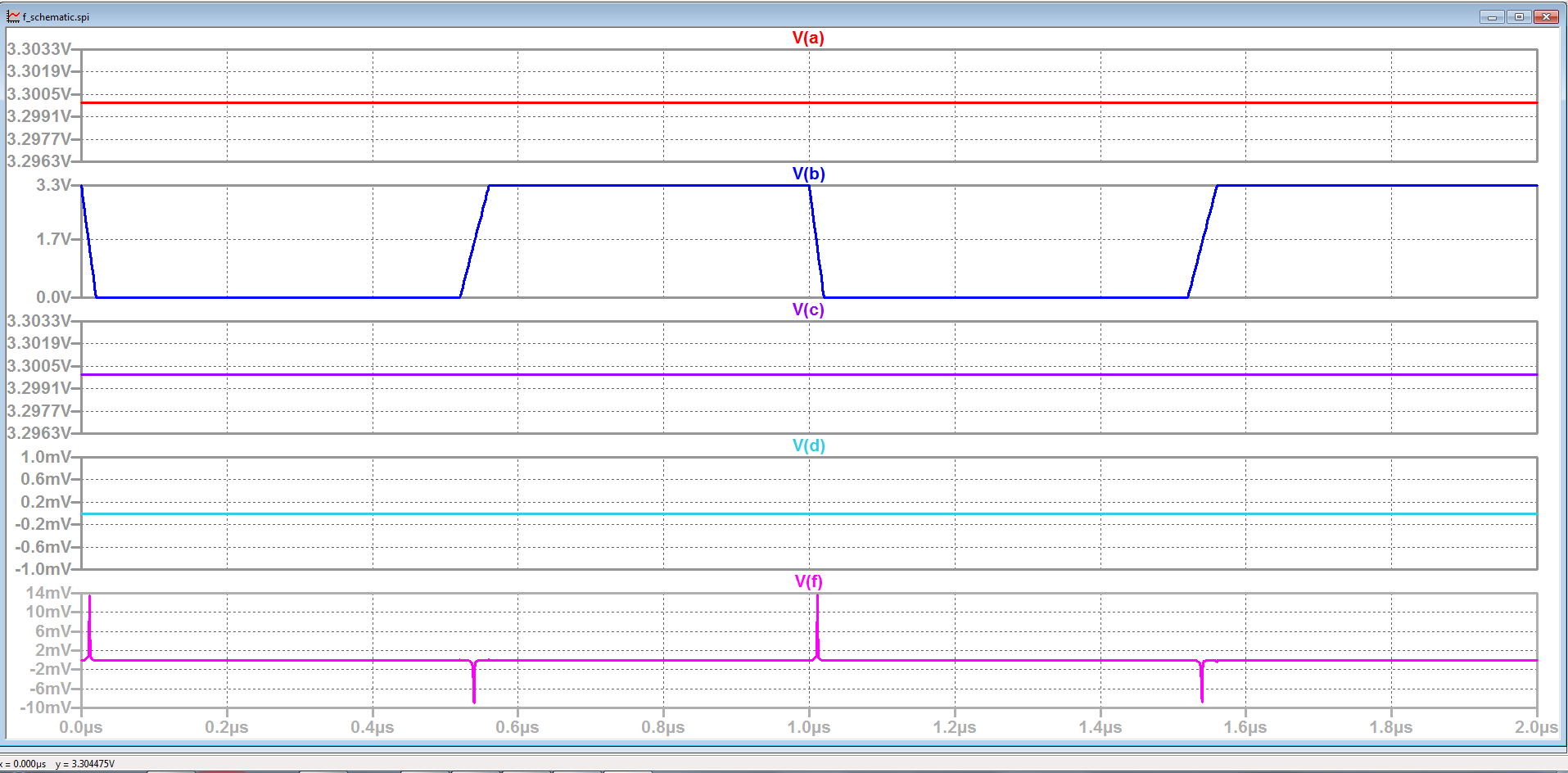
*Figure 10: LTSpice Output for Every Possible Input*

Once we have our output, we compare it to our outputs for our truth table and we can see that there two high outputs as expected in our LTSpice simulation output, when **(A,B,C,D) = {(1,1,0,1), (1,1,1,1)}**. Now that we can confirm that the circuit was designed properly to give us the correct output, we can send in our second set of inputs which we can see from our Spice Code in *Figure 11*.



*Figure 11: Spice Code for Given Inputs*

When we run our Spice Code on LTSpice, we expect to get an output of 0 as discussed previously. From our simulation seen in *Figure 12*, we can see that the output is exactly as we expect it to be.

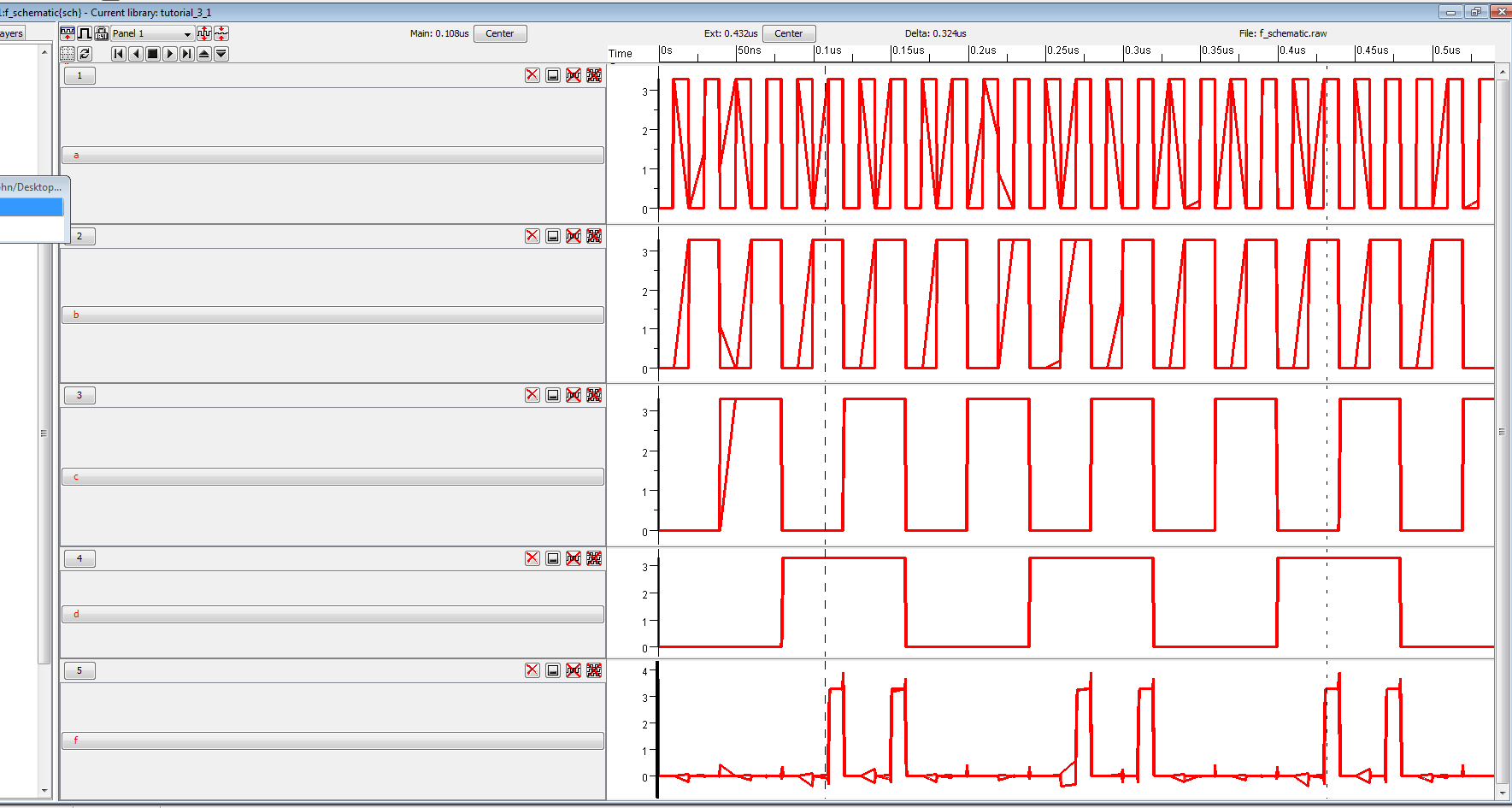


*Figure 12: LTSpice Output for Given Inputs*

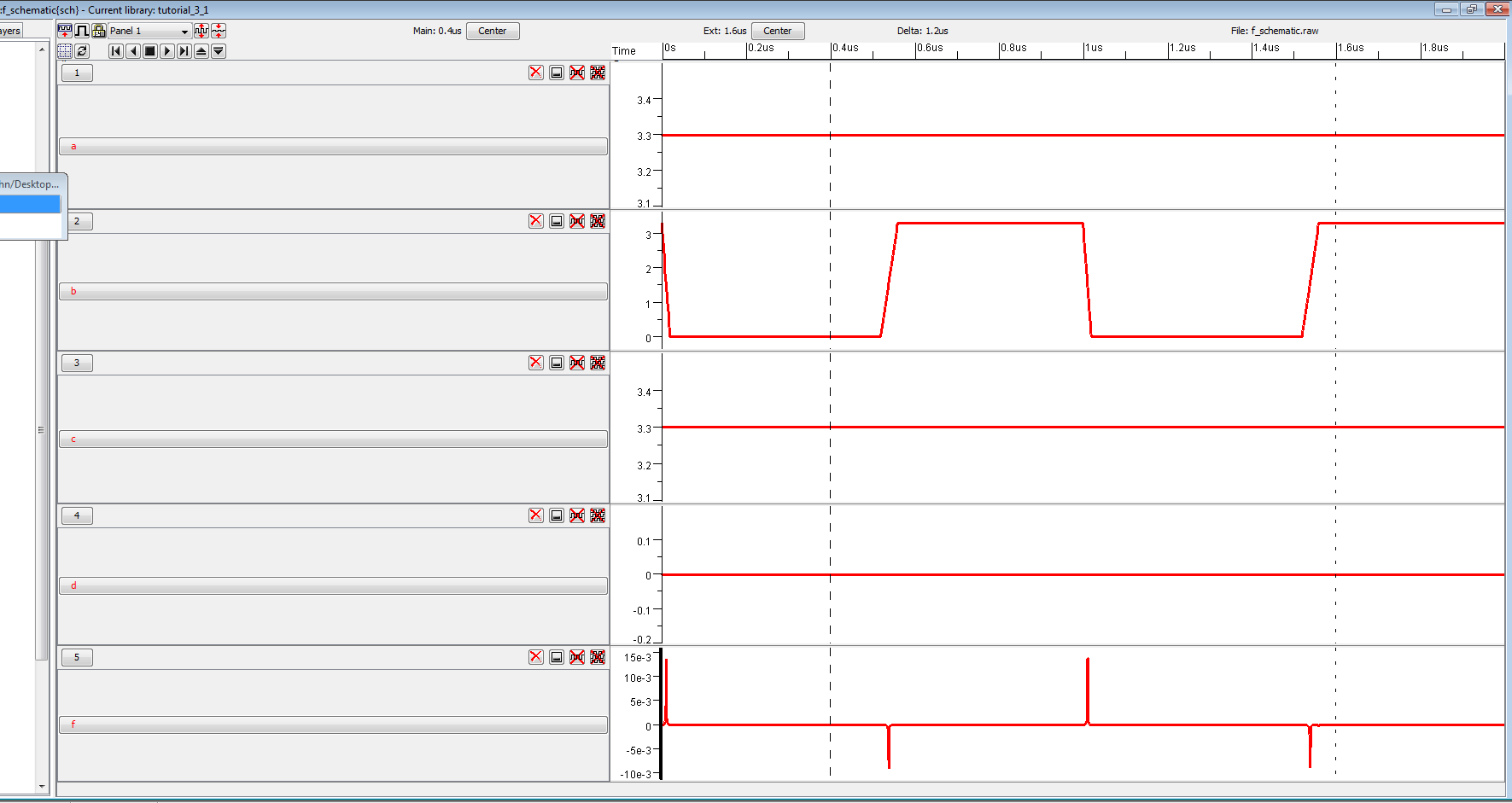
From our output, we can see that there’s minor spikes in our output but those spikes wouldn’t cause a high output since our spikes only max out at 14 mV.

**IRSIM for Electric Schematic**

Now that we can verify that our LTSpice simulations are correct, we also need to check our IRSIM simulations to see if we get the same results. When we simulate using IRSIM, we get the figure seen in *Figure 13* and *Figure 14.*



*Figure 13: IRSIM Simulation for Every Possible Input*

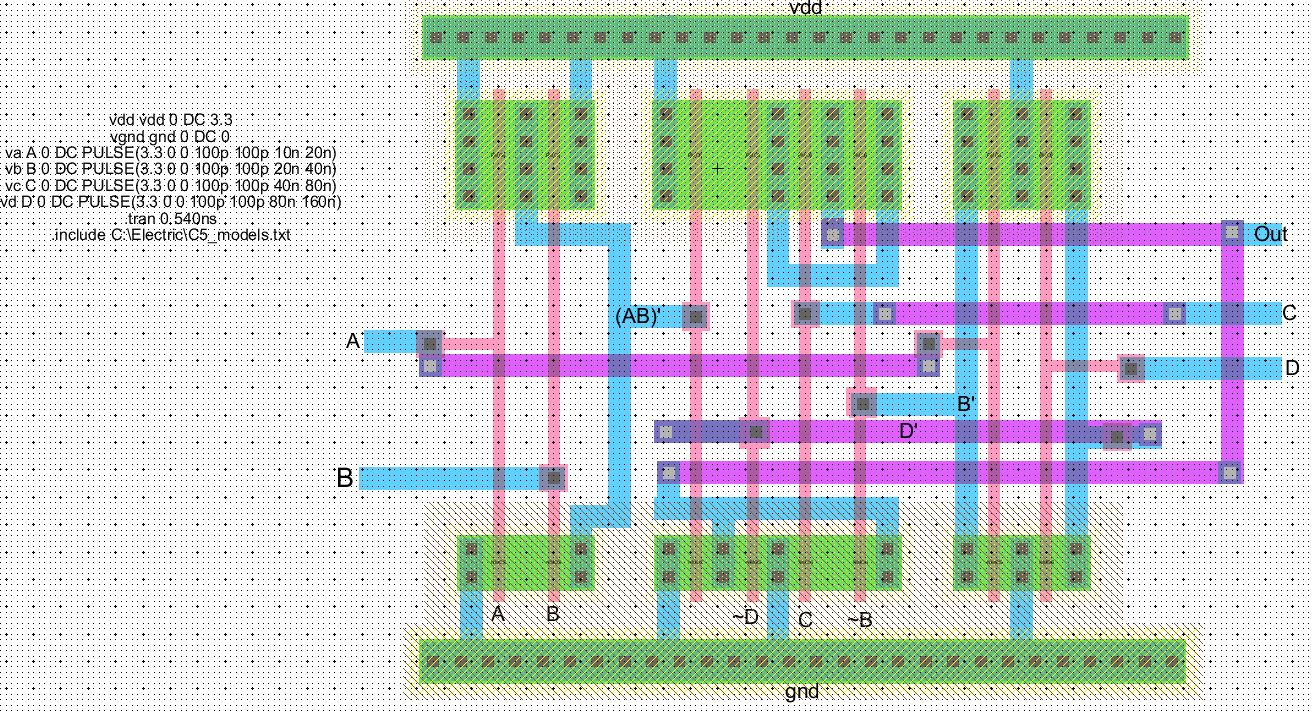


*Figure 14: IRSIM Simulation for Given Input*

If we compare our simulations for IRSIM and LTSpice, we can see that we get the same set of outputs per period for both sets of inputs.

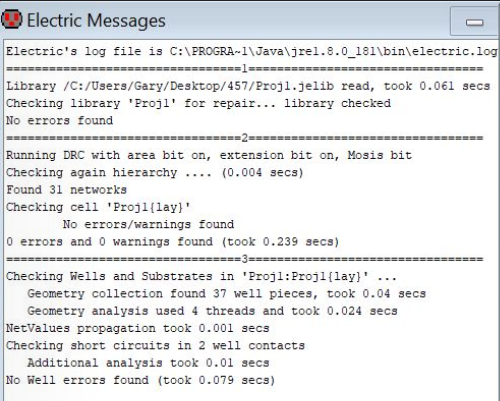
**Electric Layout for Given Function**

Once we have our schematic for our function, we move on to making our layout and to make our layout design easier, we use our Euler’s path that we created previously. As specified before, the design was made using the path **(AB)’D’CB’**.



*Figure 15: Layout Design for Function*

As we can from *Figure 15*, the design of the layout has the order of **(AB)’D’CB’** as shown in the layout at the bottom. Our three parts of our design is incorporated into the layout as well; NAND gate, two inverters and function **G**. Once we have our layout design, we need to make sure that there aren’t any errors in the design which we use the same functionality that we used for our schematic and in addition to that, we also need to do a well check to make sure that our wells are connected properly.

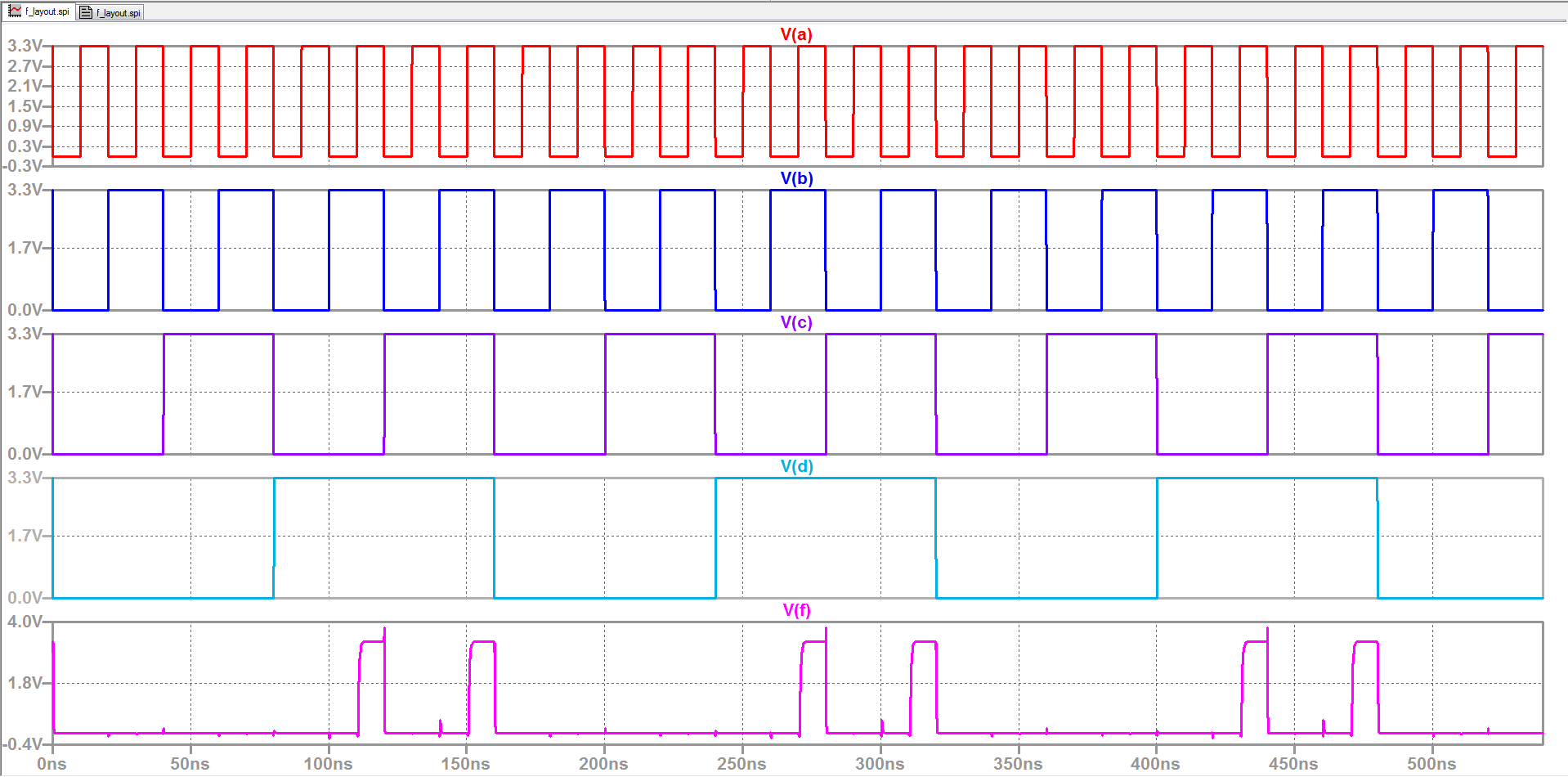


*Figure 16: Error and Well Check for Layout*

From *Figure 16*, we can see that the layout passes both the error check and the well check which means our design doesn’t violate any design rules. Now that the design has passed the design checks, we move on to simulate our layout using LTSpice and IRSIM as we did for our schematic.

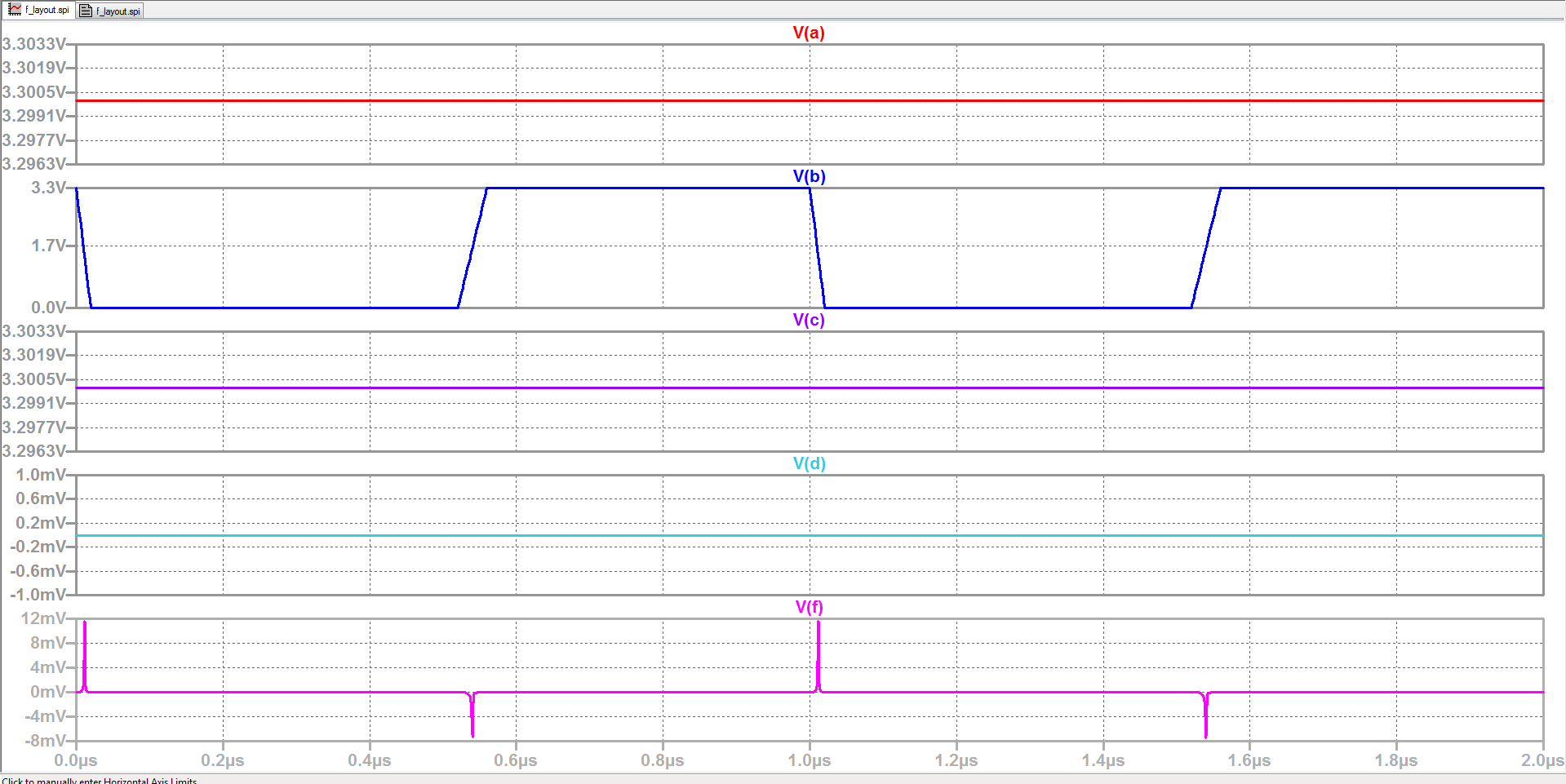
**LTSpice Simulation for Electric Layout**

The simulation of the layout would be done with two sets of inputs as well to make sure that the design works for every possible input rather than just the given inputs. Once we simulate our layout, we get the image seen in *Figure 17.*

****

*Figure 17: LTSpice Simulation of Electric Layout*

The output for the first set of inputs on LTSpice gives us the same output that we would expect with two high outputs per period. If we compare it to our LTSpice simulation of the schematic, we get the same output for the first set of inputs which means the layout was designed properly. Once we can confirm that our layout is designed properly with the same output as both the schematic output and the truth table, we can test our second set of inputs, **(A,B,C,D) = (1,PULSE,1,0).** Once we simulate our layout for the second set of inputs, we get the image seen in *Figure 18.*

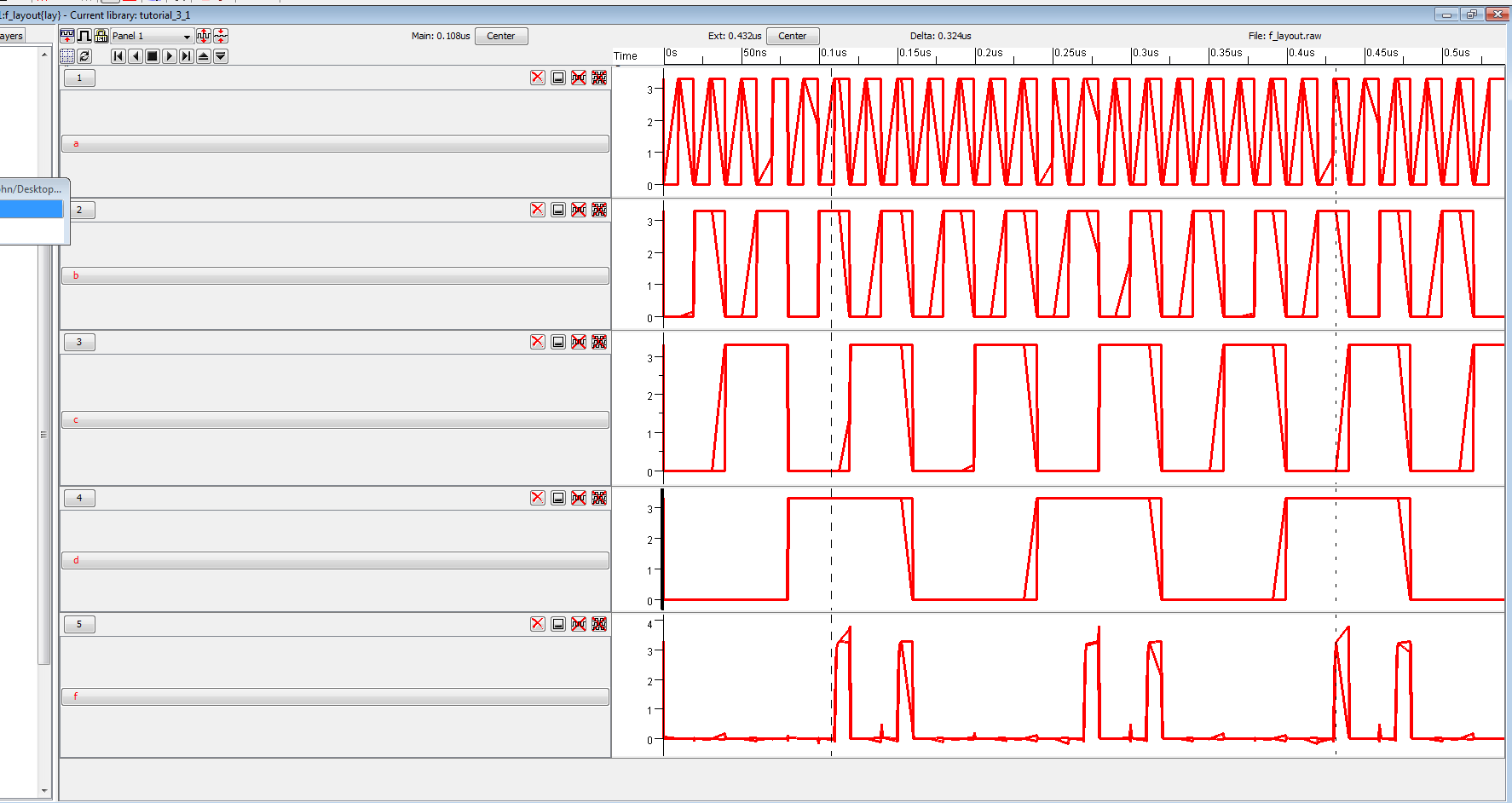
****

*Figure 18: LTSpice Simulation for Given Inputs*

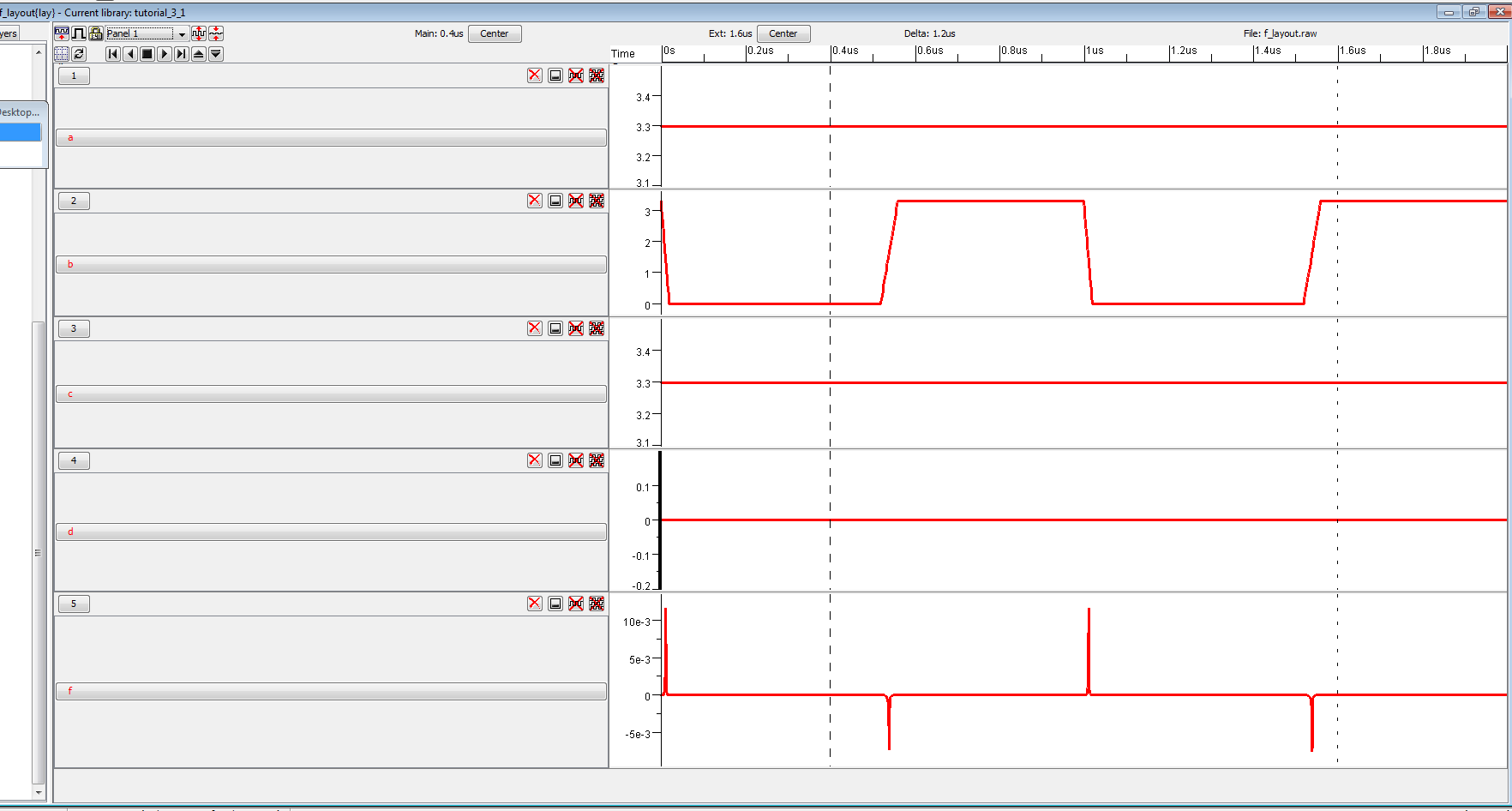
The output of our simulation is also as expected to be near 0V and to compare it to our schematic output, we can see that the output is the same which means that our schematic matches with our layout. We now move on to our IRSIM simulations which we should expect to be the same as our outputs for our LTSpice and IRSIM simulations.

**IRSIM Simulation for Electric Layout**

The output of our IRSIM simulation can be seen in *Figure 19* and *Figure 20.*



*Figure 19: IRSIM Simulation for Electric Layout*



*Figure 20: IRSIM Simulation for Given Inputs*

We can see that the outputs match our outputs from the previous IRSIM and LTSpice simulations which means that we have successfully designed the circuitry for an IC chip with our schematic and layout both outputting the desired results.

**Conclusion**

The design of complex logic functions in ICs may seem simple from what one can learn from textbooks but when it comes to designing actual functions, it is much more complicated because one must combine a series of knowledge learned in order to create the functions. With the given function **((AB)’+(B’C)+D’)’**, it makes the textbook complex logic functions as if an elementary school book to a high school textbook. It isn’t a function that you can spend a couple hours to figure out but would require many hours of thought and research to solve.

The design of the function consists of our knowledge of creating simple complex functions with the design of a NAND gate in order to get our final resulting design. One needs to remember that PUN and PDN must behave in duality otherwise, the design would not turn out to be the way it is expected. The utilization of the many resources that is provided to us is necessary to simplify the process of our design and as we have observed in our design, the digital IC mainly cares about whether it is a high or low compared to that of an analog circuit. An IC may not always be as simple as one block of circuitry but in our case, it is separated into three blocks of circuitry.

Overall, this project turned out to be a bit more of a challenge than once thought to be which is mentally stimulating and sparks the interest of the topic. The complication required to make a simple IC is considered a short duration compared to the duration required for a much more complicated circuit that can be found in industry grade designs which may require a team of engineers in order to design such as the everyday devices that we use like our smart phone or our laptops which consists of a combination of ICs.

**References**

[1] K. C. Smith, A. S. Sedra, “CMOS Digital Logic Circuits” in Microelectronic Circuits, 7th ed. New York,: Oxford University Press, 2015, ch. 14, sec. 14.1, 14.2, 14.3 pp. 1090-1114

[2] K. C. Smith, A. S. Sedra, “MOS Field-Effect Transistors (MOSFETs)” in Microelectronic Circuits, 7th ed. New York,: Oxford University Press, 2015, ch. 5, sec. 5.1, pp. 248-263