LAB 1 - Advanced Logic Design: 83511

Names & IDs:

Arad Shapira –

Tomer Hershku –

Part 1:

Briefly describe:

a. The term HDL: HDL stands for Hardware Description language. There are 3 main HDL Languages that are in use in the industry – Verilog, SystemVerilog, VHDL .

HDL describe logical hardware blocks and circuits. It specifies how digital components like gates, flip-flops, and interconnections should function. By using HDL we can do simulations, verification tasks, and synthesis of the design before the backend process .

b. The main differences between HDL and software languages like C, Python

etc. :

1. In HDL the "code" runs and read in parallel – different blocks are running on the same time (for example always combinatorial block runs in parallel to the sequential always block)

In contrast to other software languages where the reading of the code by the compiler is from the top to the bottom, line by line in a serial form.

2. HDLs describe logical blocks and circuits , other software languages are used for software purposes and are running on the CPU in the computer.

3. HDLs focuses on the describing of digital blocks and circuits, where other software languages focus on data structures, algorithms without dealing with the hardware at low level .

c. The main differences between ASIC and FPGA.

The main differences between ASIC and FPGA are:

FPGA is Field Programmable Gate Array . FPGA is manufactured on an asic processand can be programmed, and by that we can change the connectiosn between the excited hardware components inside the FPGA.

We don't need FAB for changing the logic of the FPGA, we need only to program it our needs (in contrast to ASIC, where there ASIC is implemented once and can not be changed after tape-out – changes will be done in the FAB)

ASIC – Application Specific Integrated Circuit , as its name – the given chip will be made for a specific need. After manufacturing it can not be changed or programmed.

A mistake \ error in FPGA can mostly be fixed by changing the HDL code, in contrast to ASICs where every mistake can cost a lot and won't be easy to fix.

In addition, as was said in the lecture:

FPGA vs ASIC

- FPGA is commonly used for:
  - Non-massive production chips
  - No tight performance requirements.
  - Product prototype

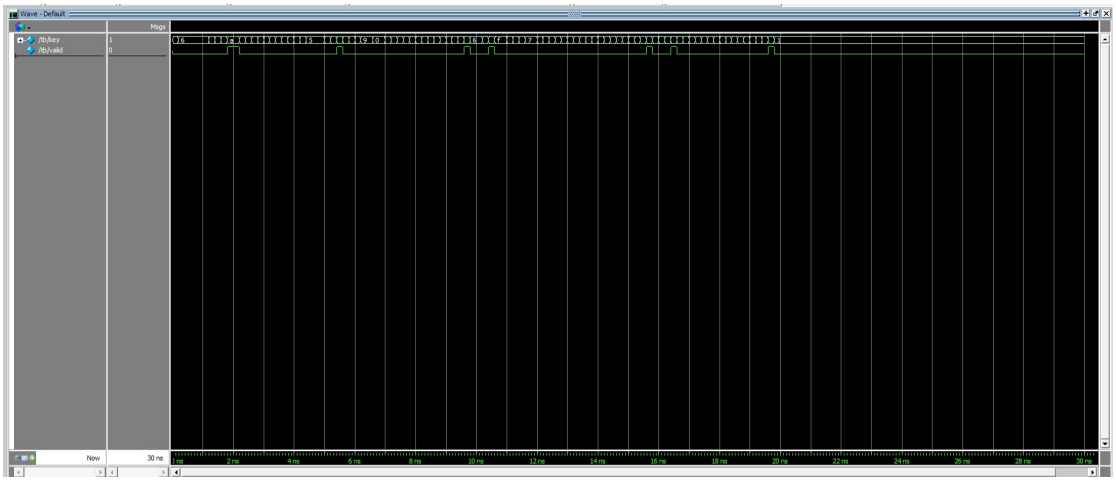
- ASIC is used for:
  - High performance (speed, power, area)
  - Massive production.

	FPGA	ASIC
Time to market	Fast	Slow
NRE (Non Recurring Engineering)	Low	High
Design flow	Simple	Complex
Unit cost	High	Low
Performance	Medium	High
Power consumption	High	Low
Unit size	Medium	Low

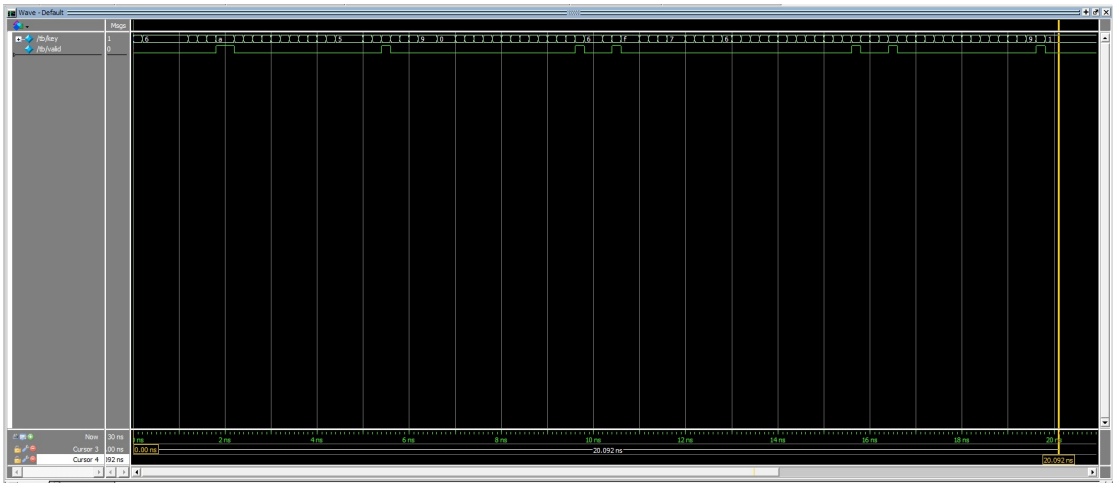
Part II: RTL Simulation:

a.

running the simulation for 30ns :



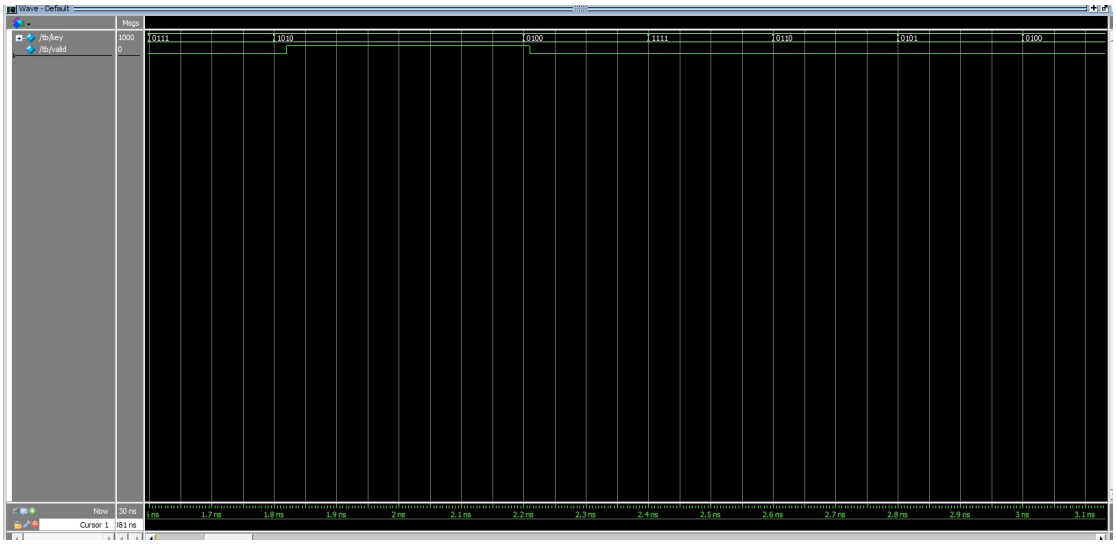
Zoom in on non “0” values :



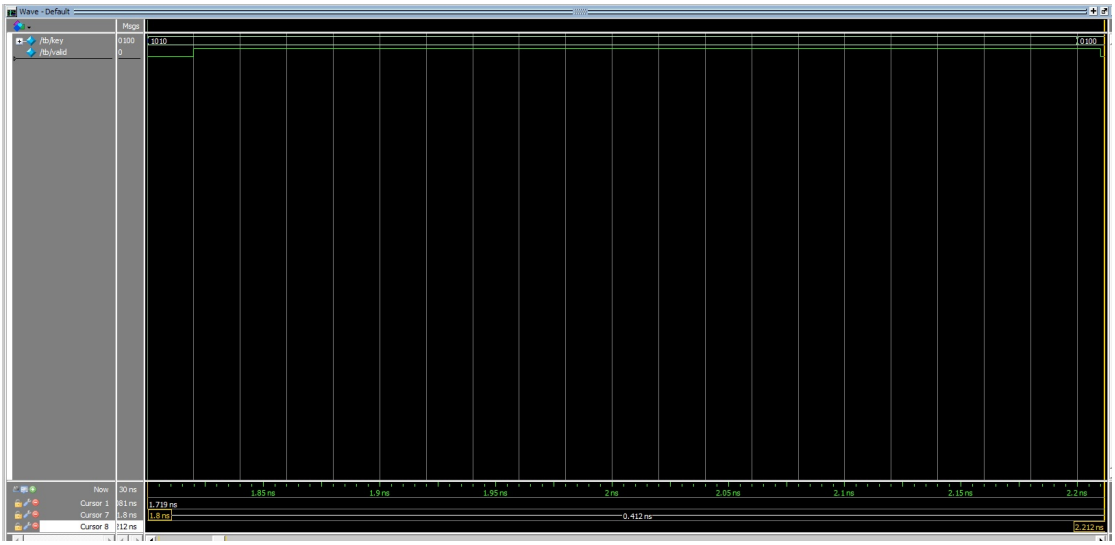
Part 3: Gate level simulation

3_a. *Measure with the cursors how much time it takes to non-glitched valid signal to de-assert. Export the waveform image.*

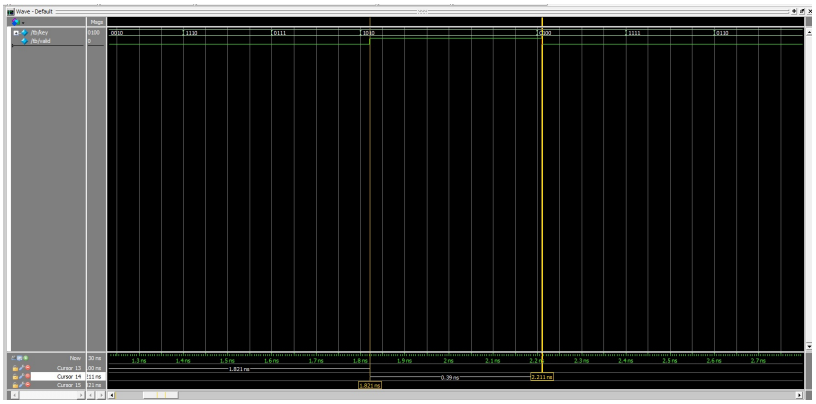
As can be seen, on ~1.8 ns in the simulation, the valid turns to “1” and then to “0”:



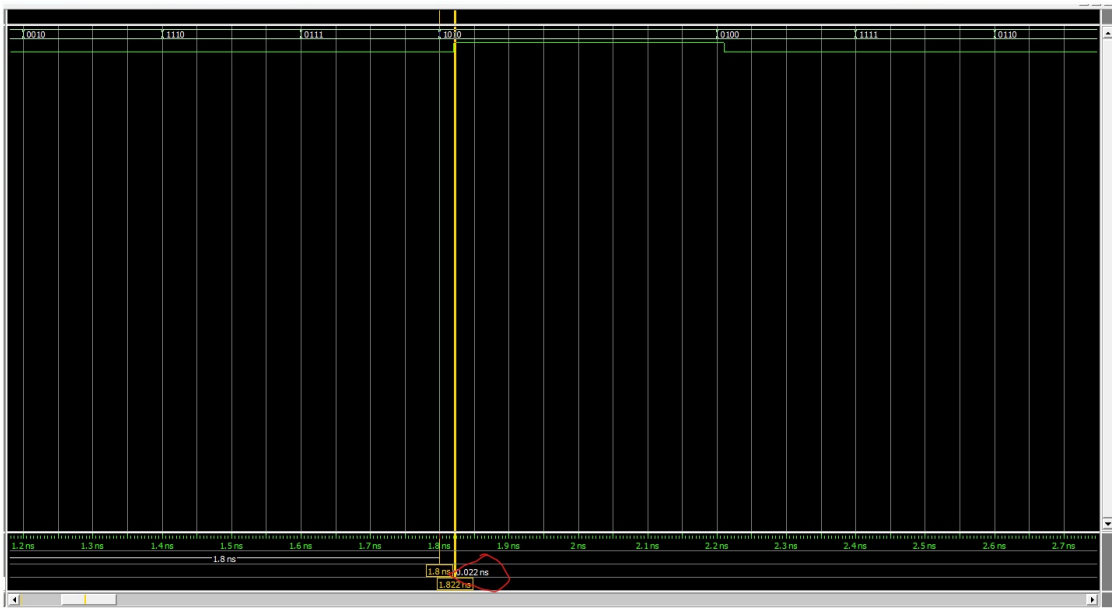
Measuring the propagation delay:



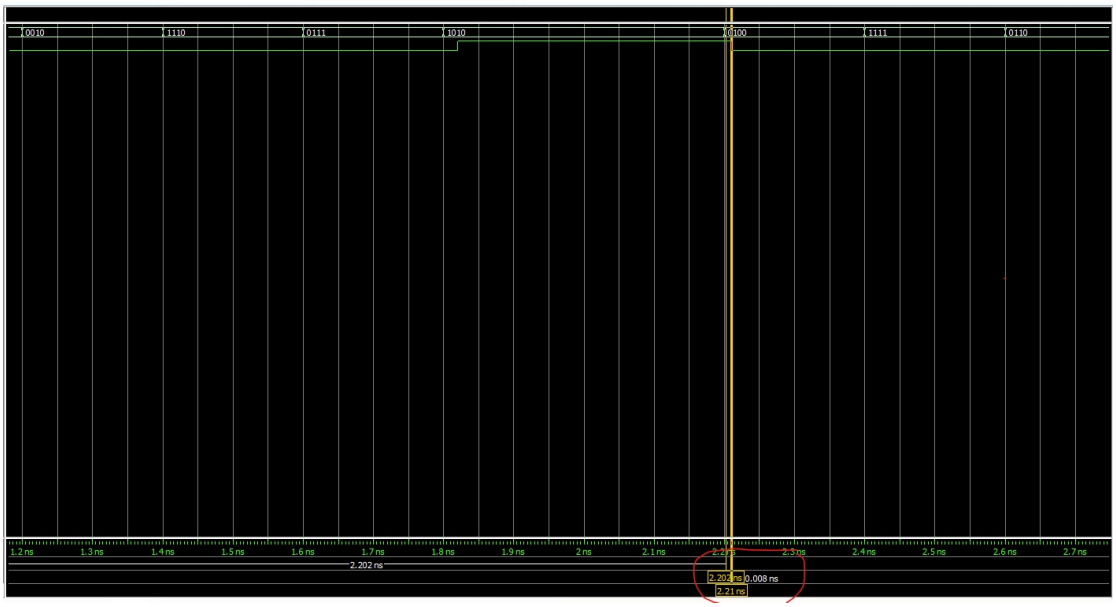
As can be seen the valid is “1” for ~0.39 ns



In addition, as it seen in the wave diagram: the time it takes for valid to change between “0” to “1” is ~0.022 ns:

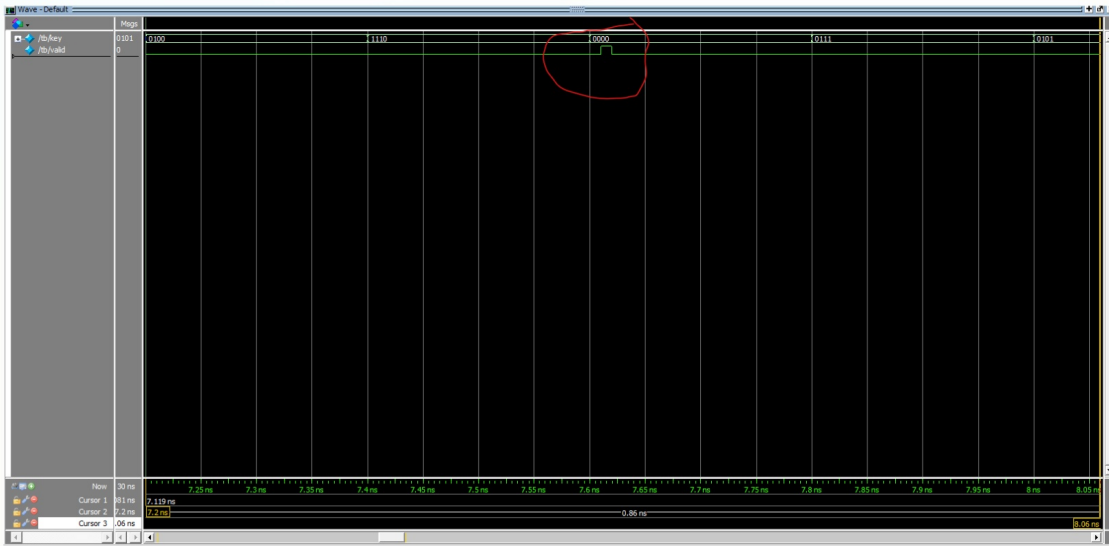


the time it takes for valid to change between “1” to “0” is ~0.008 ns:

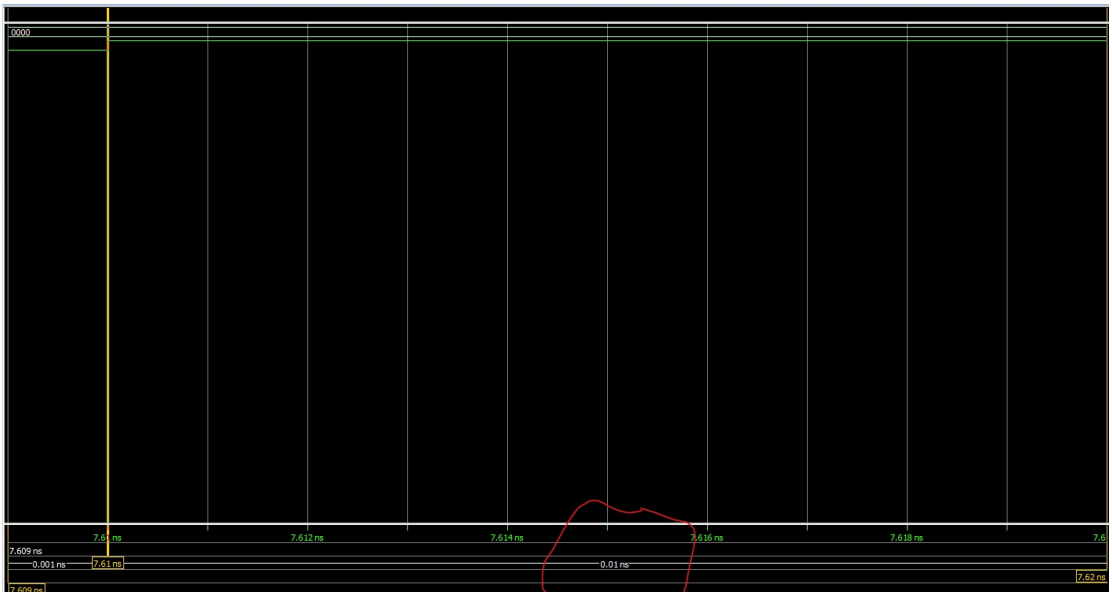


3_b : *Find a glitch on the valid signal. Measure the glitch width with the cursors and export the waveform image.*

glitch found:



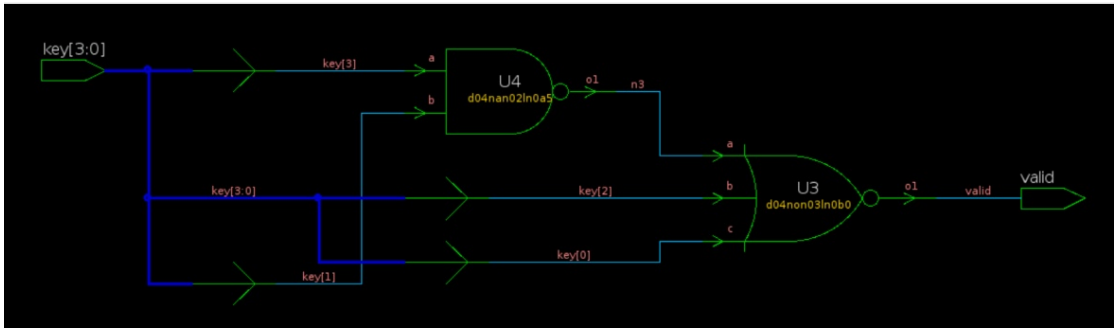
As can be seen – the glitch width is ~0.01 ns



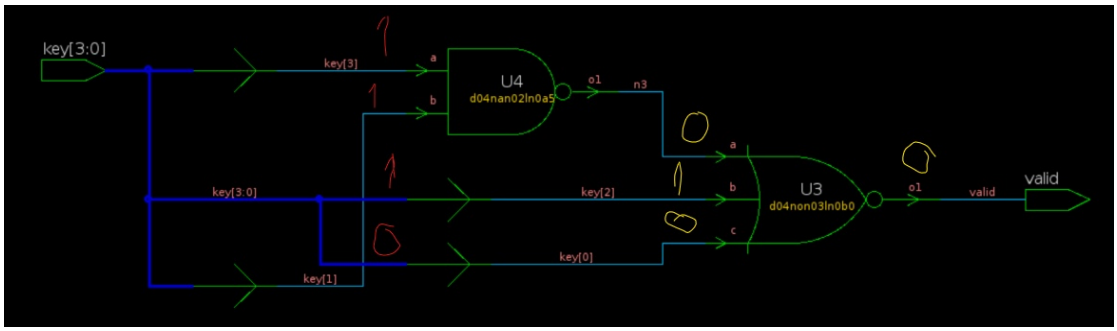
3_c: *Explain the reason for the glitch:*

The glitch occurred because for very short time the key was “1010” ,resulting raising valid to “1” . The transient of key is from “1110” to “0000”

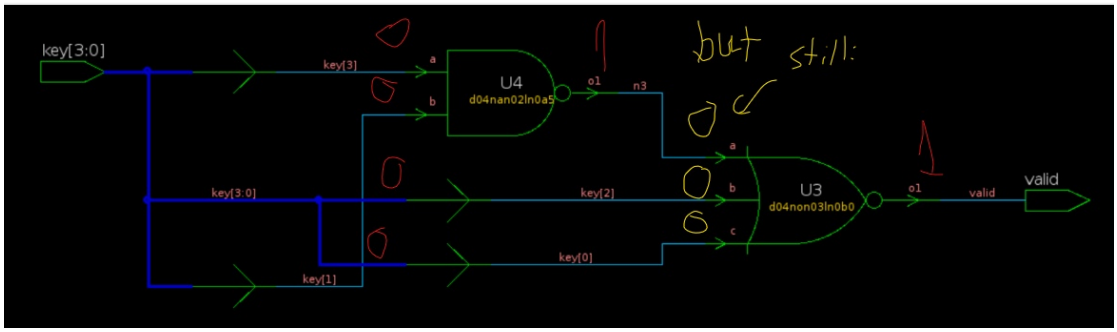
Let’s look at the schematic:



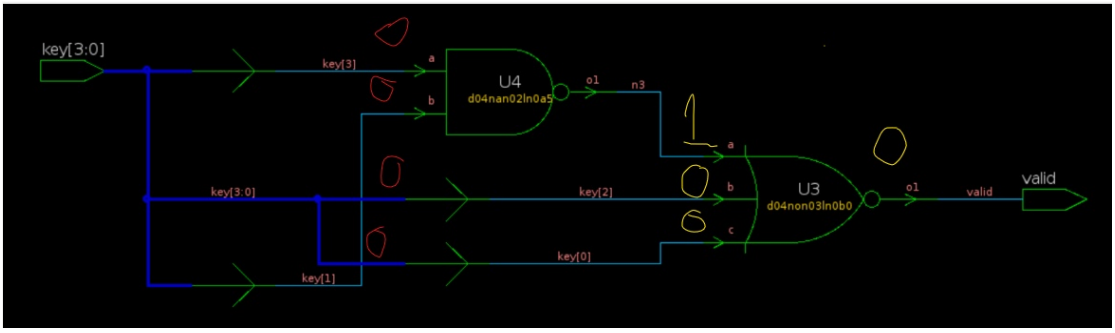
When key = “1110” – the outputs are on the screenshots:



Then we are changing key to “0000”. Because there no logic on wire key[2] the inputs to the nor_3 (U3) are “0,0,0” for short time, causing valid to be set to “1”, the output of nand_2 (U4) is “1” ,which haven’t arrived yet because of the delay of the gate.



After short time, the output of the nand_2 (U4) is “1” and it gets to nor_3 (U3), the output is “0” and the glitch is over



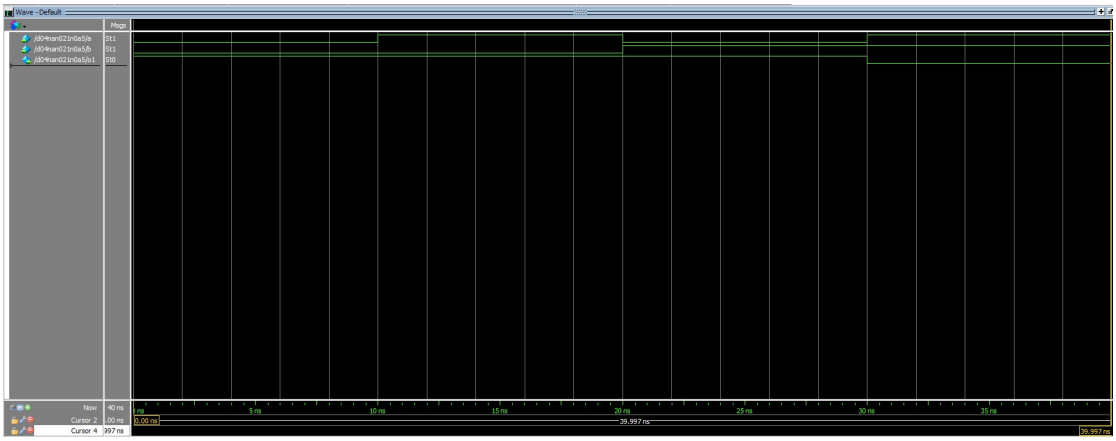
3_d: *The U4 module has 2 inputs and 1 output. Load them to the waves and list down its truth table. Which logic gate U4 represents? Export the waveform image.*

U4 is nand gate with 2 inputs and 1 output.

The truth table is:

Input_a (key[3])	Input_b (key[1])	Output
0	0	1
1	0	1
0	1	1
1	1	0

We will see all the options in the wave diagram:



We can see the output changed according to the inputs.

U4 represents nor gate with 3 inputs, 1 output .

3_e: *Write a do file that do the following steps: • Restart the simulation. • Add the key and valid signals to the wave. • Run the simulation for 30ns while after 10ns from the beginning of the simulation the n3 signal is forced to '1' for 5ns only.*

Arad solution:

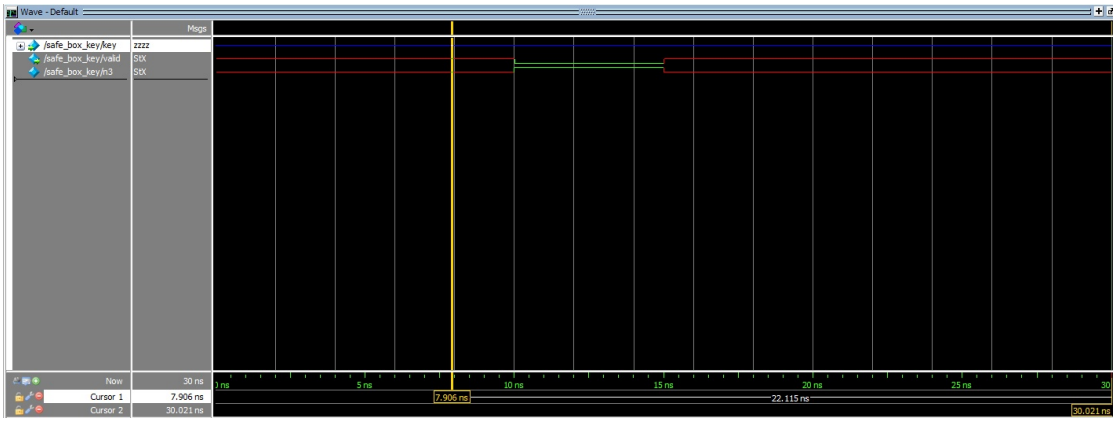
We have created a do file:

```
vlog safe_box_key_gl.sv
vsim
add wave key
add wave valid
add wave n3
force -freeze n3 1 {@ 10 ns} -cancel {@15 ns}
run 30 ns
```

And run through the shell:

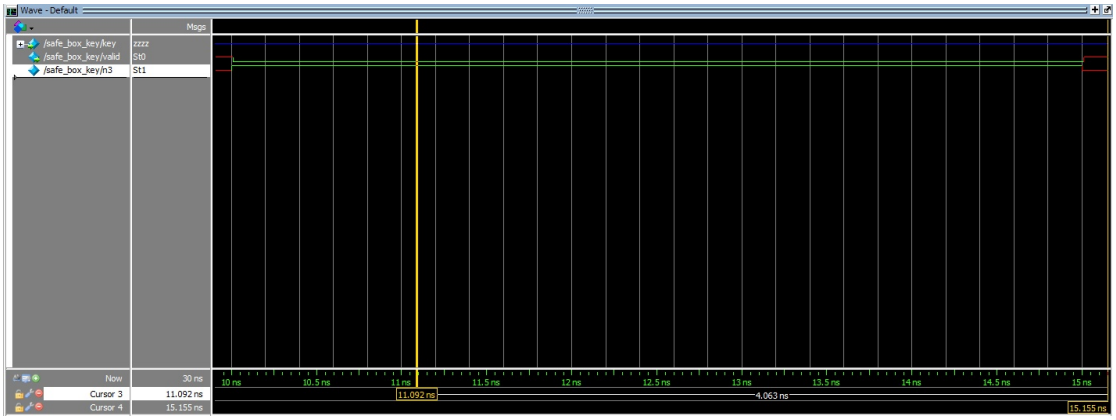
```
VSIM(paused)> do 3_e.do
```

And we got the results:



As we can see, there's no digital data except of 10 - 15 ns in the simulation.

Zoom in:



As can be seen – raising of n3 to “1” resulted to valid (= the output of the module) to be raised to “1”. We could not have concluded it – gate level modules are hard to logically understand.