Fall 2019

California State University, Northridge

Department of Electrical & Computer Engineering



Lab Experiment 2

*Writing VHDL Testbenches Using File IO*

Juan Silva & Tumen Khureltulga

(Group 16)

September 24, 2019

ECE 524L

**Table of Contents**

**I. Introduction and Problem Statements…….………………………...…Page 2**

**II. Procedure……..…………………………………………….…………...Page 2**

**III. Testing Strategy………………………………….....………………......Page 4**

**IV. Results…………....…………………………………....…………….......Page 4**

**V. Analysis…………....……………………………....……………….……..Page 6**

**VI. Appendix……………...………………………....………………….…...Page 6**

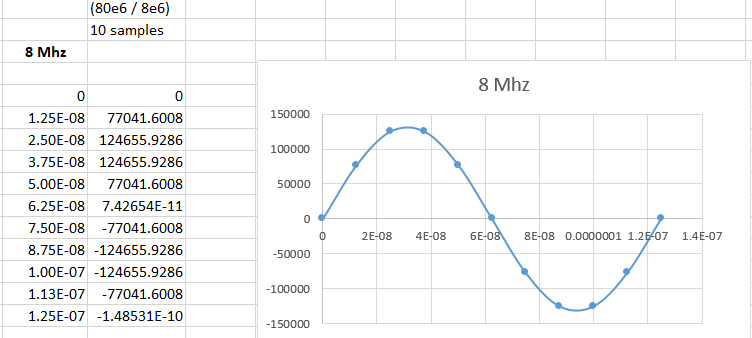
1. **Introduction and Problem Statements**

In this experiment, VHDL code for a parametrized CIC filter is provided and a testbench must be designed to verify the functionality of the source code. This filter contains an integrator and a combinational section that run on varying frequencies. These components receive a sine wave input at a specified frequency and the designed CIC filter will output the possible stopband and passband frequencies.. Therefore, the testbench must initialize the CIC filter, quantize a sine wave input, and export the output onto a text file. Three frequency cases will be tested for the parametrized CIC filter and output to a text file to view the output data.

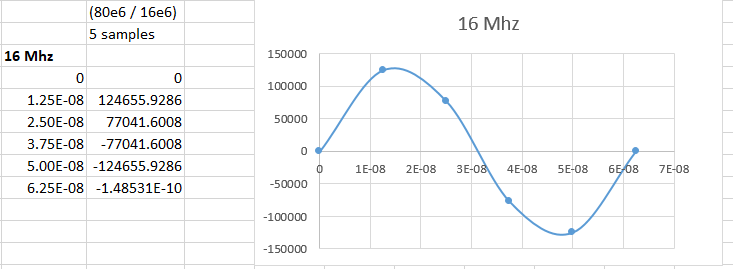
1. **Procedure**

The CIC filter code is added as a top level design and a test bench is designed to verify that the filter is working. To quantize a sine wave at 8 Mhz, an Excel sheet table is created and displayed as points in a graph shown in *Figure 1.1*. Column 1 represents the delay for each point being read. This delay is represented by the Zedboard frequency (100 Mhz) divided by the input frequency (8 Mhz) resulting in 12.5 ns. Column 2 represents a point on the graph represented by the equation in *Equation 1.1*. If the graph generates a sine wave, then a process is created and the data from the table is converted to an 18-bit signed logic vector.  
 The number of cycles for each frequency input is determined by the ratio between the integrator (80 Mhz) and the combinational section (16 Mhz) resulting in a 5:1 decimator ratio. Once completed, the output data is exported into a spreadsheet and plotted for verification. For the 8 Mhz passband, the filter will pass a sine wave with some change in magnitude. For the 16 Mhz case, the filter will suppress the input sine wave. For the 24 Mhz case, the filter will pass the sine wave with some varying amplitude.

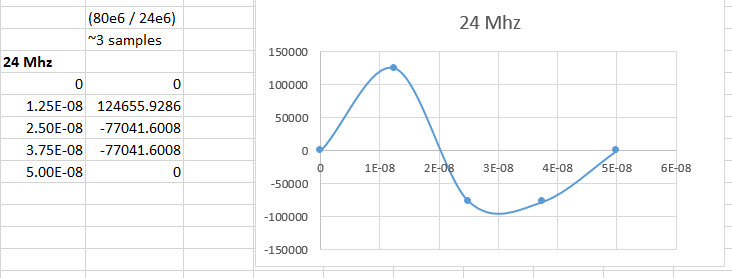
**Equation 1.1 -** Mathematical representation of input frequency.

****

**Figure 1.1 -** 8 Mhz Sine Wave input



**Figure 1.2 -** 16 Mhz Sine Wave input

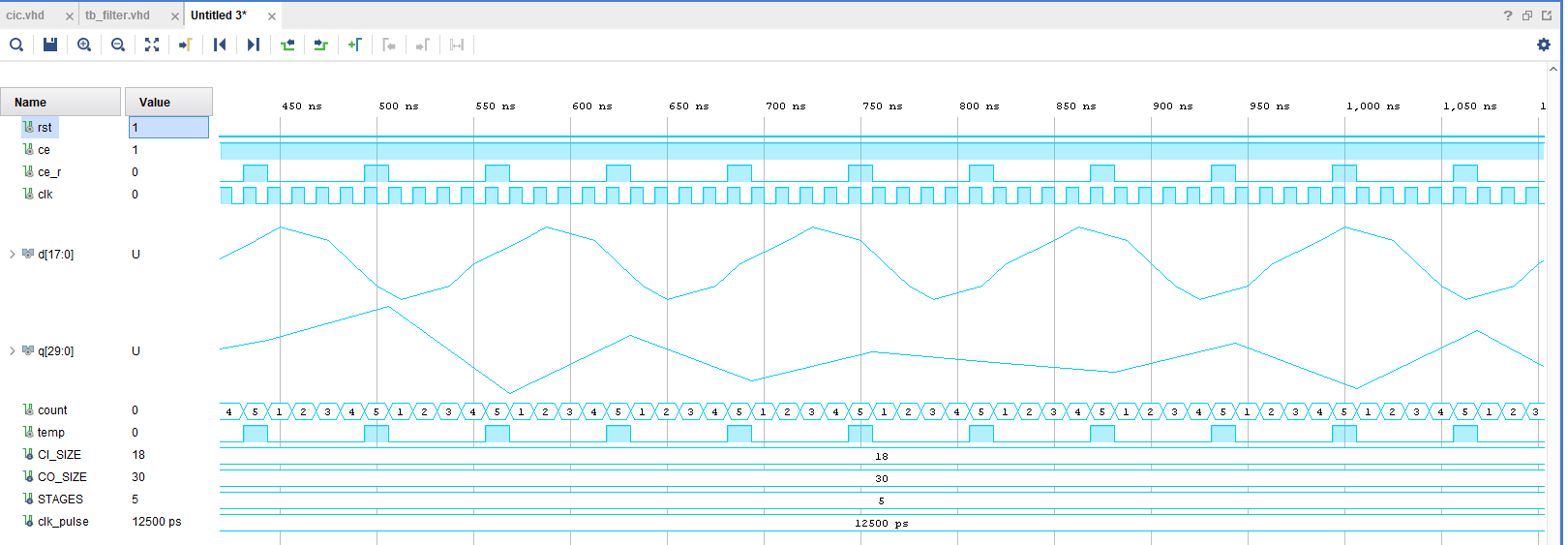
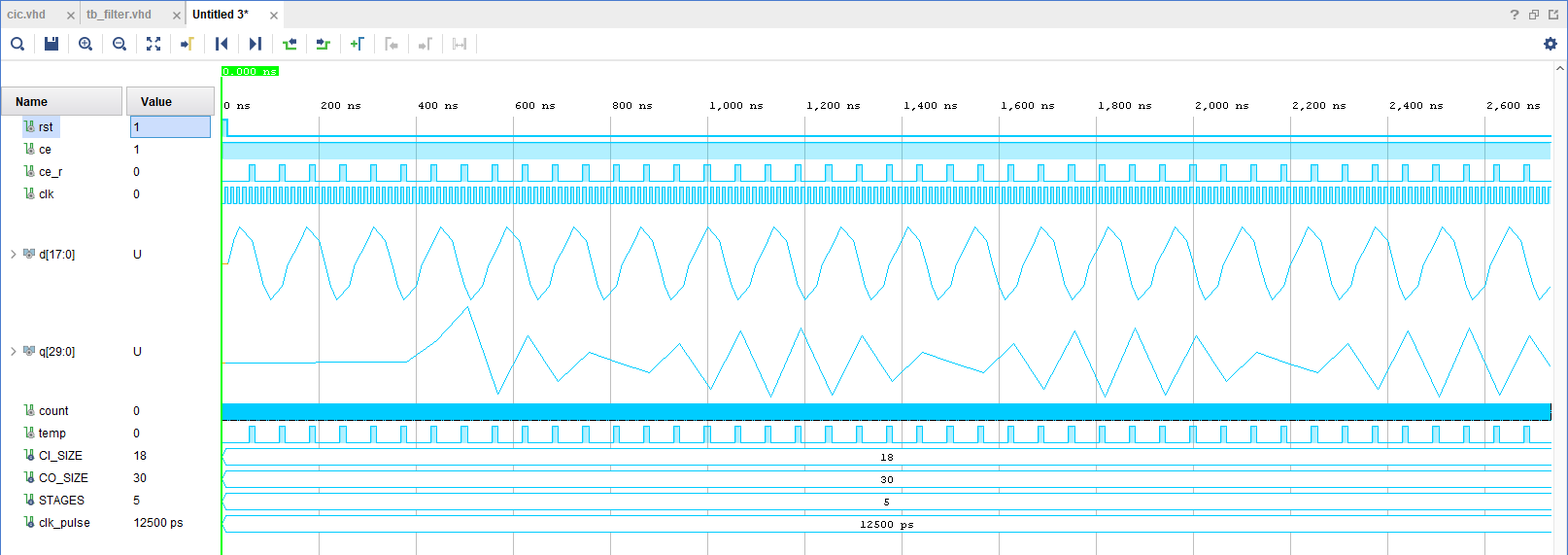


**Figure 1.3 -** 24 Mhz Sine Wave input

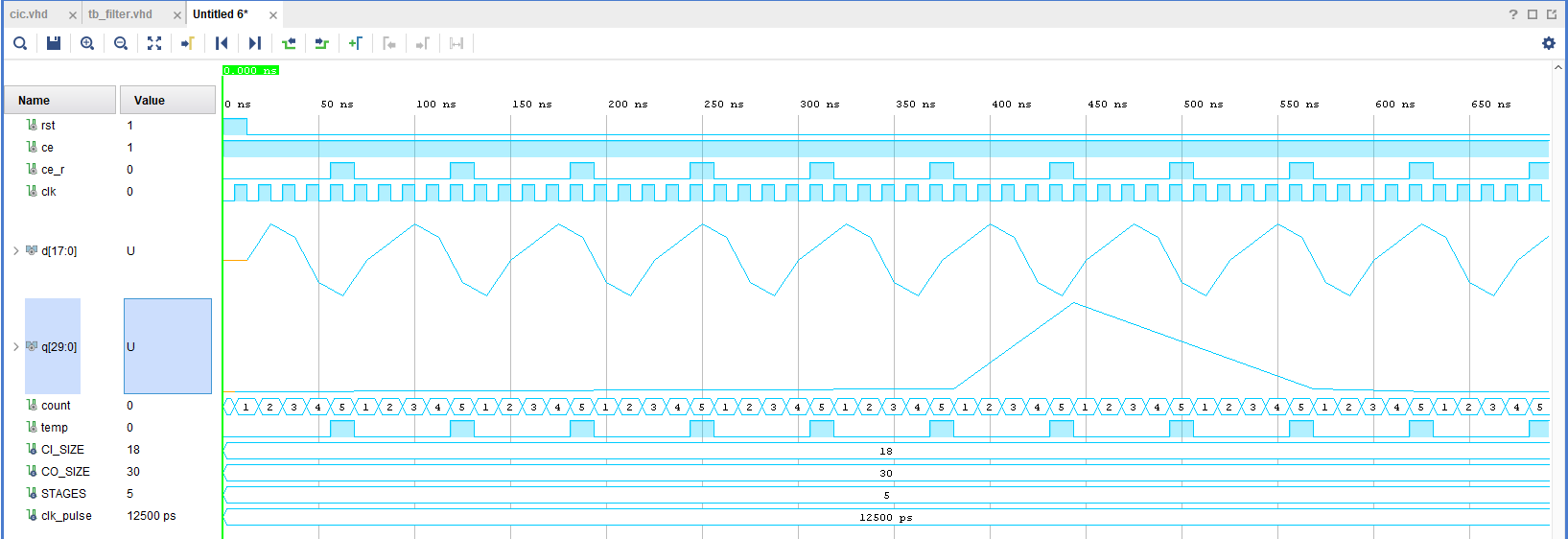
1. **Testing Strategy**

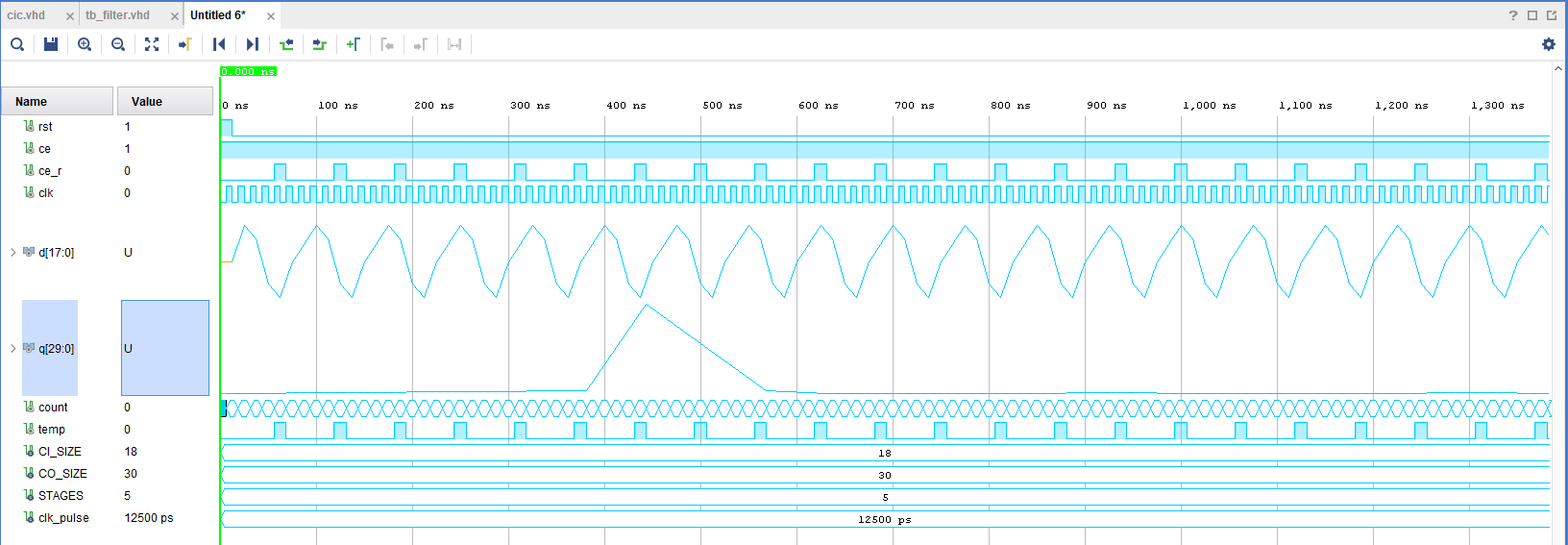
The CIC filter code is carefully examined to see how it works. Afterwards, the testbench is broken down into various parts. First, we check for a test stimulus in order to see if the component responds to input. This includes testing the clock, reset, clock enable and reading the d inputs. In parallel, we check if the quantized data generates a sine wave. This is done using Vivado software tools such as the waveform viewer. Changing the input data display to Analog and Signed Decimal converts the digital signal into an analog sine wave. This is shown on *Figure 1.2* and onward. The Vivado software tools were used extensively to check if the decimator worked correctly. The waveform viewer is beneficial to sync the decimator to the clock every five clock cycles.

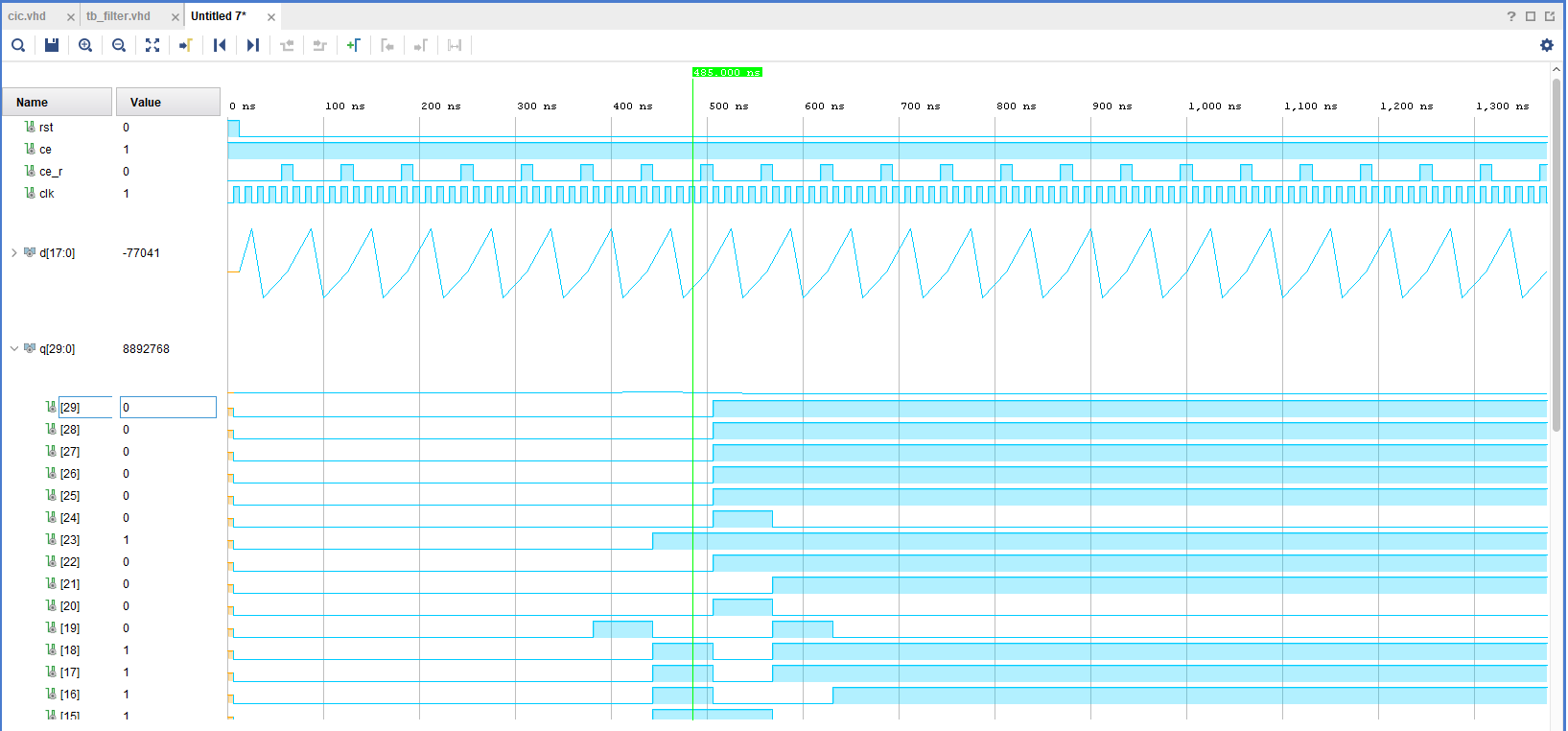
1. **Results & Data**

 **Figure 1.1** - 8 Mhz Passband Frequency Waveform (1 cycle)  
  


**Figure 1.2** - 8 Mhz Passband Frequency Waveform (3 cycles)



**Figure 1.3** - 16 Mhz Passband Frequency Waveform (1 cycle)  
  


**Figure 1.4** - 16 Mhz Passband Frequency Waveform (3 cycles)   
 

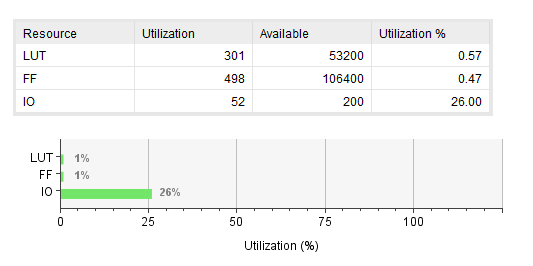
**Figure 1.5** - 24 Mhz Passband Frequency Waveform (1 cycle)

1. **Analysis**

This lab experiment is an exercise in examining and verifying source code. A quality testbench tests for all possible cases. In this case, three frequencies are quantized and tested. Additionally, this also highlights the advantages of using FPGAs for digital signal processing. For this application, analog signals are received and processed to observe the functionality of filters. To receive accurate results, the decimator must match the specified 5:1 ratio for sampling to be correct. Various errors were encountered when testing.   
 One major obstacle is avoiding the output from generating unknown values. This is due to component signals being uninitialized, misalignments in the decimator counter, an inaccurate reset pulse, or an invalid digitized signal. These errors were encountered while simulating the designed testbench. One thing to take away from this experiment is the use of external applications to assist in DSP applications.

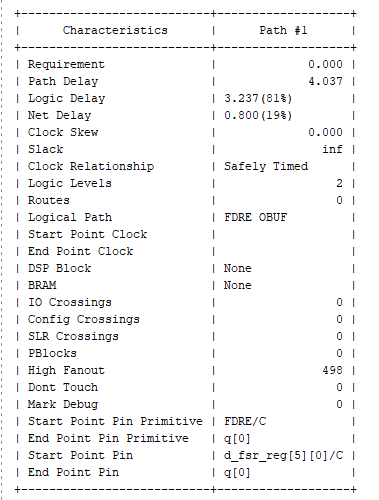
1. **Appendix**

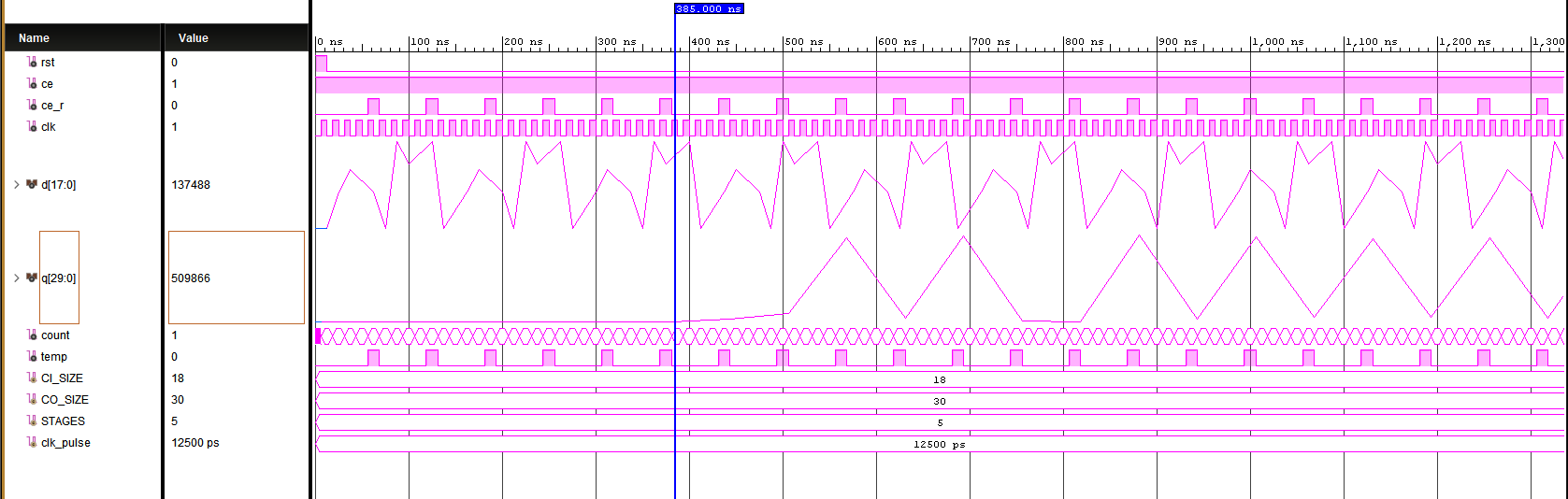
**Procedure Questions:**

1. 

The filter uses 301 LUT, 498 FFs, and 52 IOs no matter what the input is. It stayed constant for all three cases.

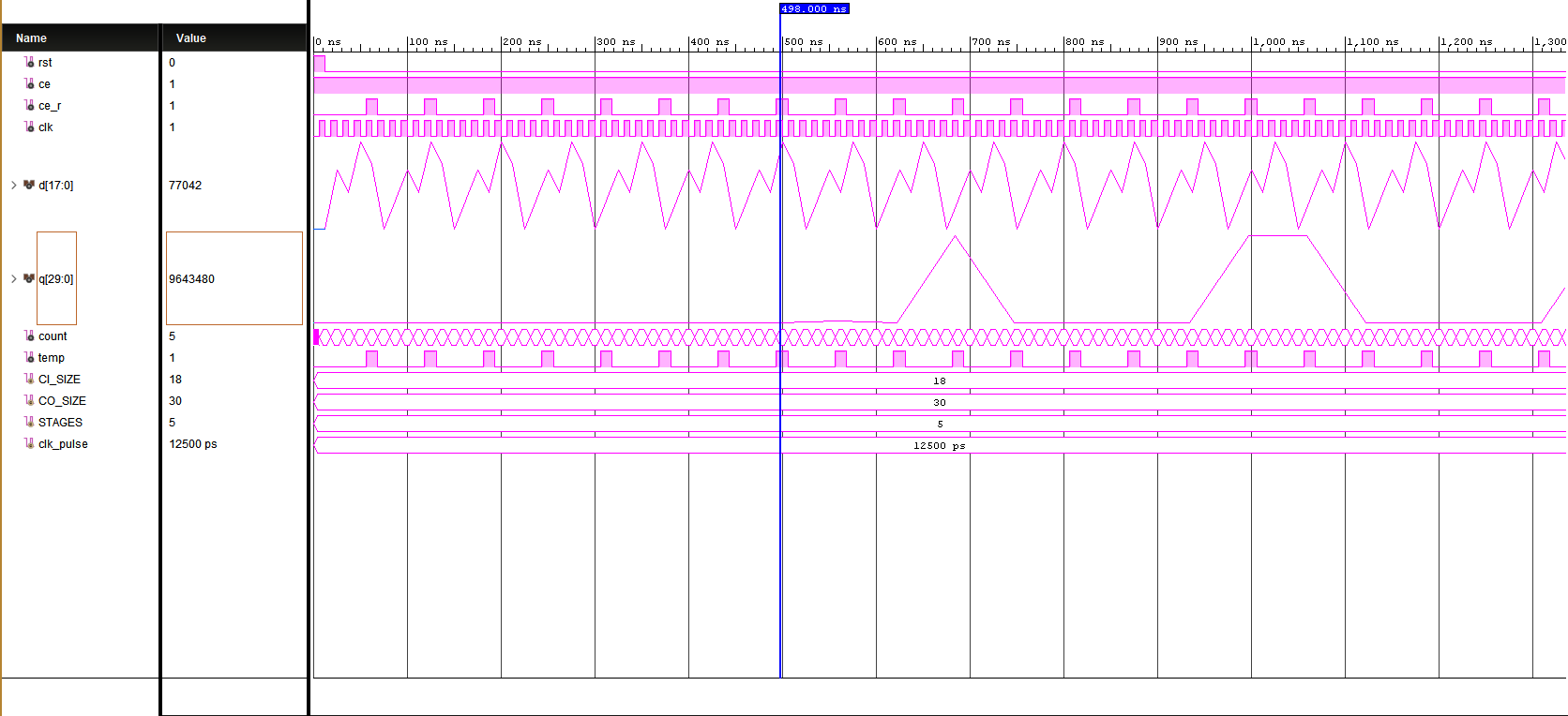
1. The critical path delay for the design for all designs inputs are 4.037 ns. This value is from running the timing simulation. Comparing this value to the value obtained from the simulated waveform, there is a difference of 2 ns. This is because timing simulation is exact compared to the behavioral simulation waveform. This difference was noticed in all three cases.





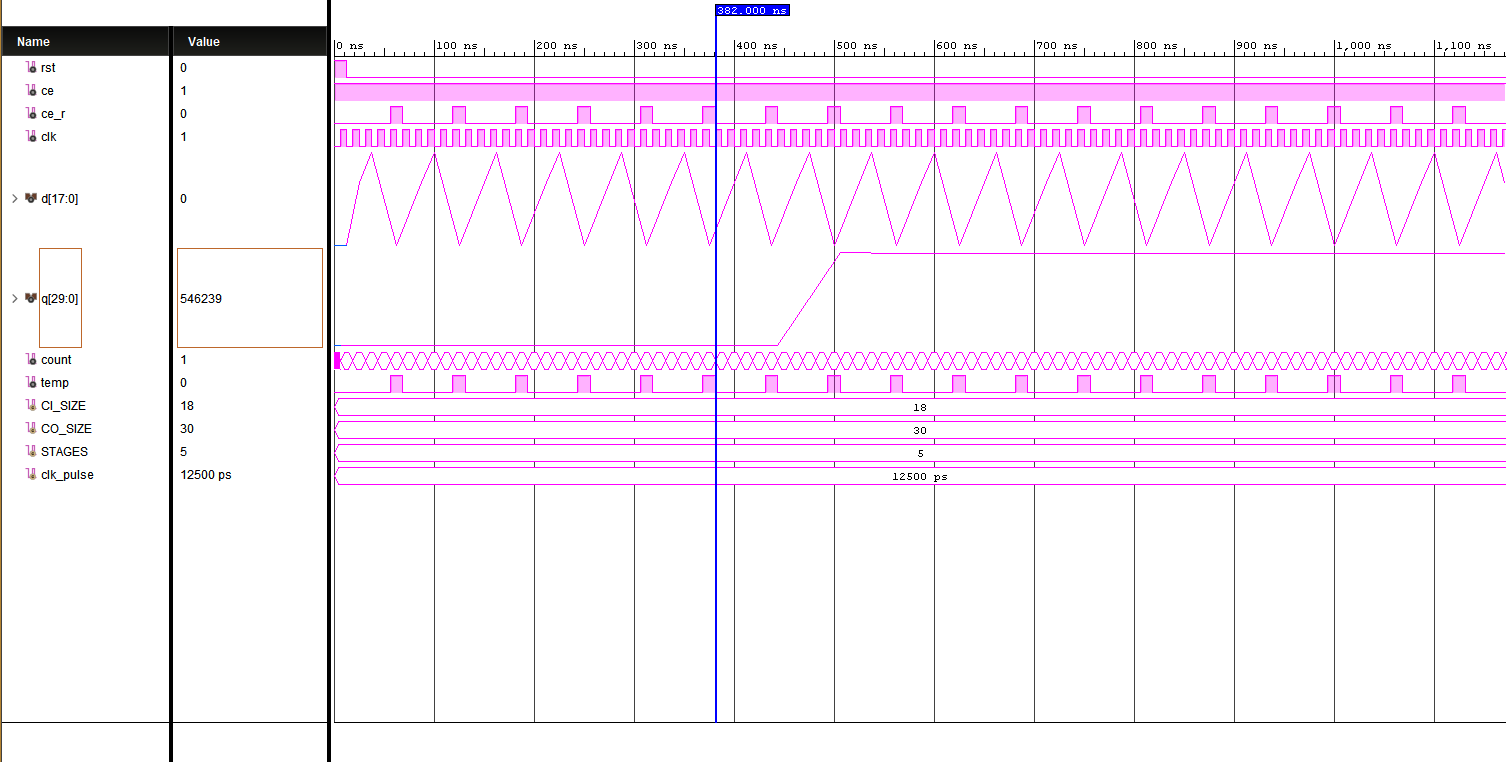
**Figure 1.6** - 8 Mhz Sine wave input

When the input is 8 MHz Sin wave, the latency is approximately 385 ns. This is about 31 clock cycles after.



**Figure 1.7 -** 16 Mhz Sine Wave input

For 16 MHz sin wave, the first output comes a little later than the 8 MHz. As seen above figure, the output changes are noticed at 498 ns. This is 40 clock cycles later than the first input to the CIC filter.



**Figure 1.8** - 24 Mhz Sine Wave input

In the case of 24 Mhz, the output changes at 382 ns. In other words, the output changes after 31 clock cycles.

The CIC filter takes at least 31 clock cycles to process the input data and outputs.