Time to Digital Convert on Xilinx’s Virtex-5 xc5vlx30 FPGA

20 May 2017

|  |  |
| --- | --- |
| Bryant Gonzaga  [bryant.gonzaga@stonybrook.edu](mailto:bryant.gonzaga@stonybrook.edu)  Stony Brook University | Victor Tellez  [victor.tellez@stonybrook.edu](mailto:victor.tellez@stonybrook.edu)  Stony Brook University |

Supervised by:

Peter Milder

[peter.milder@stonybrook.edu](mailto:peter.milder@stonybrook.edu)

Stony Brook University

# Introduction

This paper describes how a Time to Digital Converter (TDC) is created using Field Programable Gate Array (FPGA). Specifically, a Xilinx Virtex-5 xc5vlx30 FPGA. The TDC is expected to receive two pulses. The first pulse, *‘start’*, indicates when the TDC should start counting; this pulse should be at least one clock cycle long. The second pulse, *‘stop’*, indicates when the TDC should stop counting; this pulse should be at least 8 clock cycles long. The TDC is to measure the time from the rising edge of *‘start’* to the rising edge of *‘stop’* in picoseconds.

# Building Blocks of Our TDC

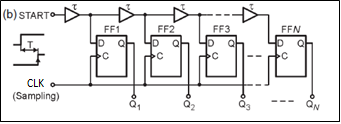
Since the aim of a TDC is to measure very short time interval, in the order of picoseconds, a basic binary counter cannot be used. A Tapped Delay Line (TDL) is used to measure when the *‘start’* and *‘stop’* signals come in. Specifically, it is used to measure how much earlier the signal comes in before the following rising edge of the clock. A regular binary counter is then used to measure how long the pulse stays high for. With these three measured times, we can use the following formula to get the final measurement:

Equation



## Tapped Delay Line

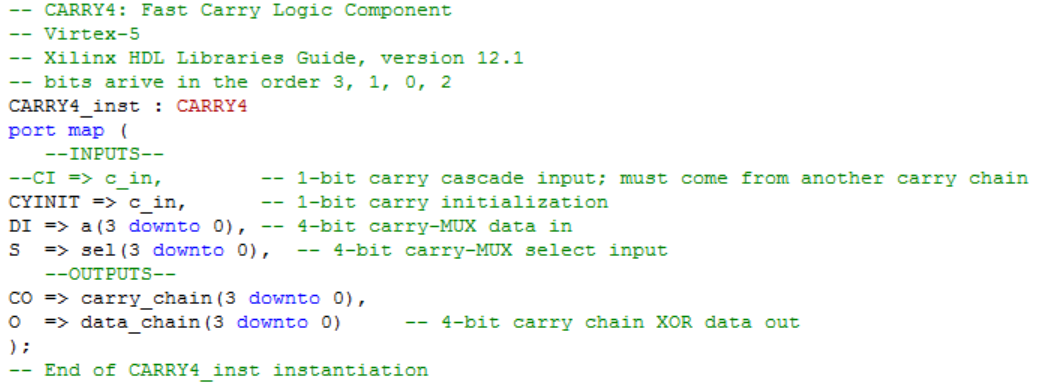
A Tapped Delay Line (TDL) is a series of buffers that are sampled at a given time *t*. Figure 1 shows an example of a TDL. Each buffer adds a delay of τ. The D flip-flops, shown in figure 1, are there to record how far the signal has propagated.



Figure

### Creating the TDL

Xilinx’s CARRY4 primitives are the basic building blocks in our TDL. The CARRY4 primitives take the place of four buffers in figure 1. Figure 2 shows how to instantiate a CARRY4 primitive in VHDL.



Figure

The CARRY4 primitive has 4 inputs. However, only 3 can be used at once. The CI and CYNIT are mutually exclusive; they cannot be used at the same time. It has 2 outputs. The CARRY4 primitives are meant to be carry-look ahead units with special routes to each other. They are usually used for speeding up arithmetic computations. By taking advantage of their special routing, we can get a very small delay between each instance. By cascading them we can make a TDL. The most significant bit of *‘CO’* is connected to the next CARRY4 instance through the *‘CIN’*.

## Course Counter

A binary counter is used as the course counter. The counter increments once every clock cycle. When taking the value of the binary counter it should be multiplied by the clock period to get *Bcountert* in equation 1.

## Hamming Weight Calculator

Hamming weight of a binary string is the number of high bits in that string. This is essential because we characterize the TDL in terms of how many bits are high. The number of high bits is directly proportional to the amount of time the signal arrived before the clock’s rising edge. The specific relationship between high bits and time is deduce later in the data section. With this relationship, we can get *startt* in equation 1.

## Calculation Unit

This unit takes the output of the pre\_tdl, pst\_tdl and crs\_cnter units as input. pre\_tdl and pst\_tdl are both instances of a TDL that take *‘start’* and *‘stop’* as the c\_in signal, respectively. The crs\_cnter is an instance of a binary counter. The calculation unit, firstly, saves the inputs at appropriate times when the data is available from the pre\_tdl, pst\_tdl, and crs\_cnter. The output of the pre\_tdl is saved to a register whose clock is triggered on the system clock only when the *‘start’* signal is high. Likewise, pst\_tdl is saved to a register whose clock is triggered on the system clock only when the *‘stop’* signal is high. Then, it counts the number of high bits in the pre\_tdl and pst\_tdl through the hamming weight module.

### Equations & Derivations

From these counts, a time can be calculated using the following equation:

Equation



First let’s consider the following base formula:

Were *‘y’* being the bit count, *‘m’* is the slope, ‘*x’* is the time offset of the signal, and *‘b’* is the y intercept.

The slope was calculated by running a simulation of eight-thousand iterations. The signal comes in seven nanoseconds before the clock rising edge and each subsequent signal comes one pico-second later. So that in the seven-thousandth iteration the signal comes in zero pico-seconds before the clock rising edge.

The data collected was in the form of a text file. The first line in the text file represented an offset of zero pico-seconds (meaning the signal arrived seven nanoseconds before the clock rising edge). The following lines represent the signal coming in one pico-second after the previous line. Each line has a three-hundred and twenty-bit number representing the TDL output. The text file had eight-thousand lines one for each of the eight-thousand iterations. Using MATLAB, the data was plotted so that the time offset was on the x-axis and the bit count was on the y-axis.

This yielded figure 9. From this data, we calculated slope to be -0.0432. The shift, as explained later, is due to the delay from the clock to the flip flops.

The y-intercept was said to be 320 because this is the total number of bits in our TDL. So therefore, we can never see a bit count greater than 320. With these numbers, we get the following formula:

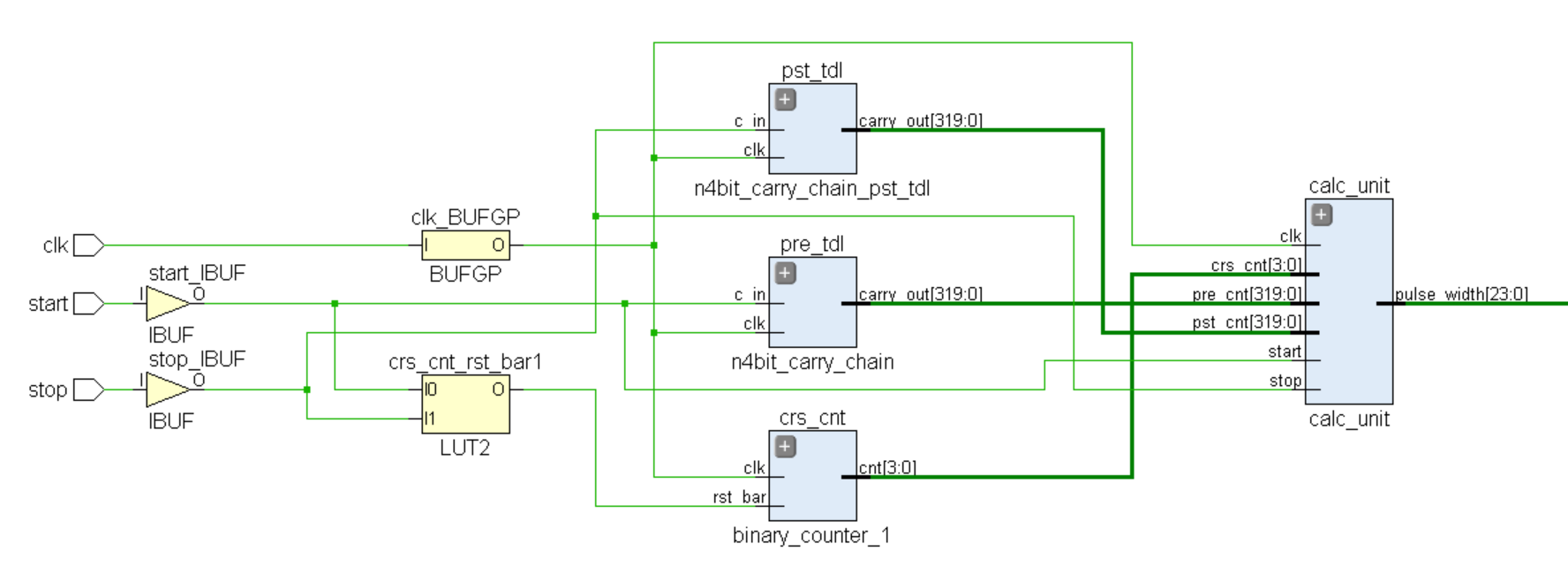
This formula is not in the form that we want it in so through some manipulation we get.

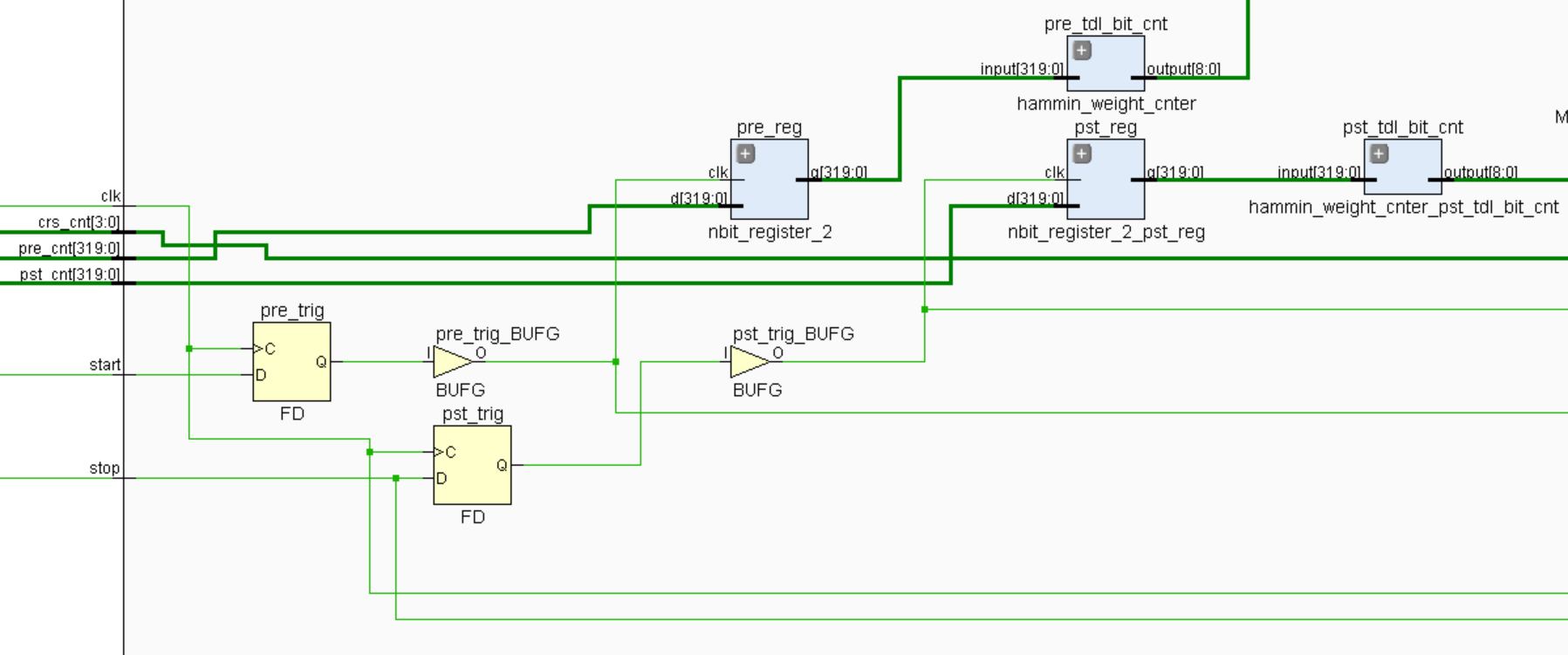
This is still not exactly what we are looking for since we do not want the *‘timeOffset’*. We want to know how much earlier the signal arrived before the rising edge. We need:

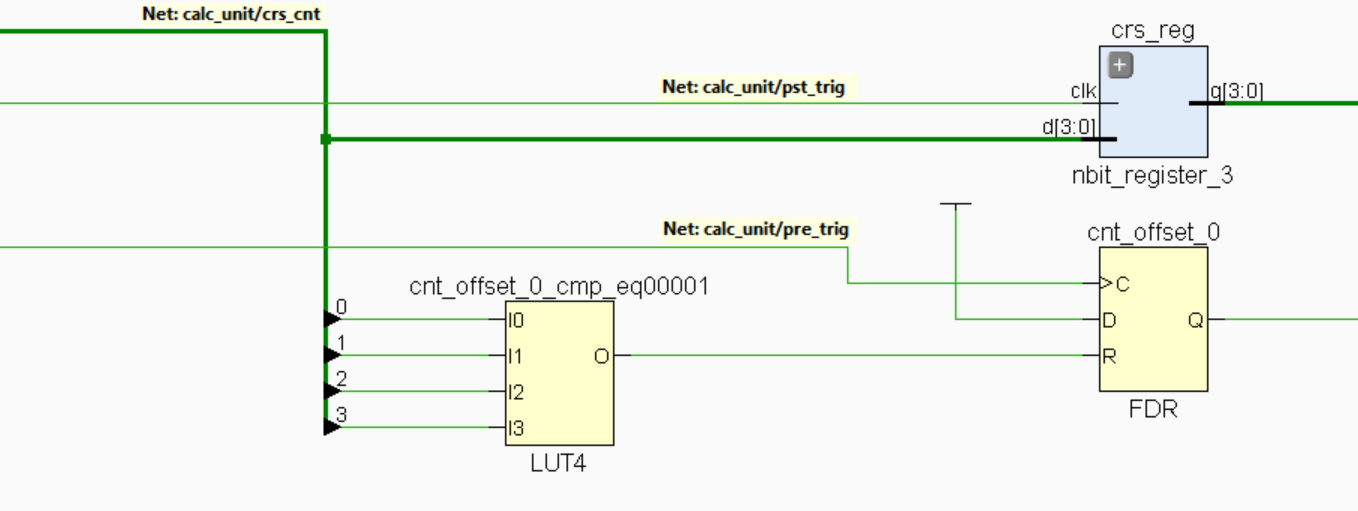
*‘C’* is the length of the TDL. In other words, it is how long the signal takes to propagate from the first CARRY4 primitive in the chain to the last CARRY4 primitive in the chain. This constant is found by observing the timing diagrams or by setting *‘bitCount’* to zero. This would mean that the TDL did not see any signal and that the offset was greater than or equal to *‘C’.* It means that the signal came after or right on the rising edge. ‘C’ is found to be or 7407.407. This matches the time given by the simulator of 7.409 nanoseconds. By plugging in equation 3C into equation 3D we get equation 2. The equation that is synthesized is as follows

With these times and the course counter time, we can now plug into equation 1 to figure out the final time.

# TDC







# Data

## Data Regarding Single TDL

### CARRY4

After running several simulations, the following information could be deduced regarding a CARRY4 primitive. The following observations where seen when the CARRY4s where cascaded together. Where the most significant bit of CO of the previous CARRY4 was connected to the following CARRY4’s CI input. Figure 3a shows a representation of this (technology schematic). All S inputs of all the CARRY4s are high. All DI inputs of all CARRY4s are low (however they are really don’t cares because of select value). The first CARRY4 instance does not follow these patterns; all following CARRY4 instances follow the same behavior. Figure 3b shows an abstracted view of what is happening inside each CARRY4 primitive. If we assume that CI is initially low and then becomes high at some time *t*, the following is the order in which the bits become asserted:

CO (0): *t* + 160 picoseconds

CO (1): *t* + 130 picoseconds

CO (2): *t* + 185 picoseconds

CO (3): *t* + 91 picoseconds

The first observation that can be made is the order in which the bits get asserted: 3 🡪 1 🡪 0 🡪 2.

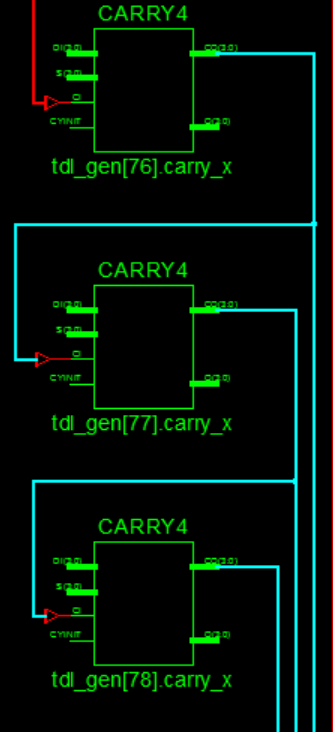
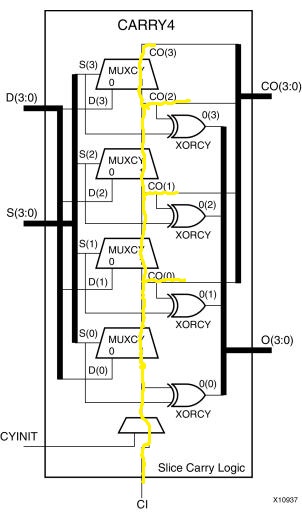


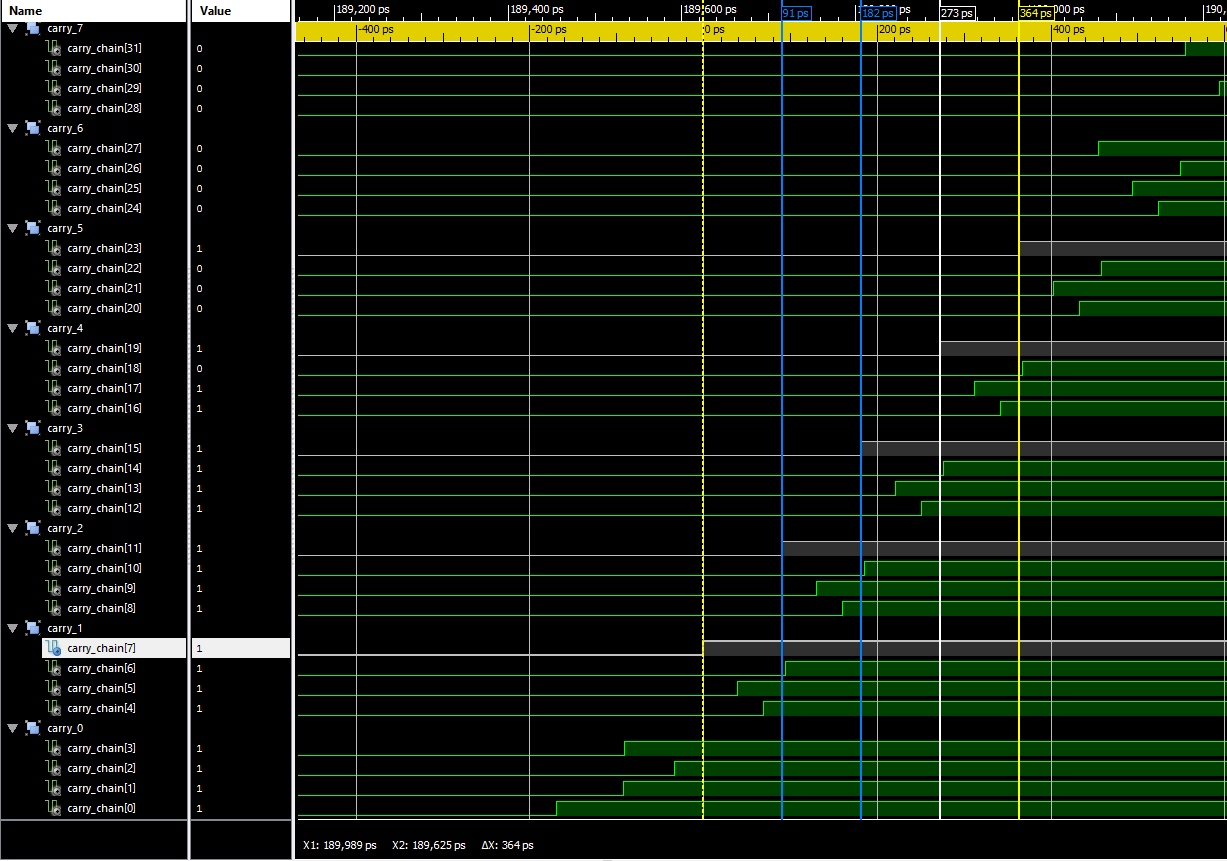
Figure 2b

Figure 2a

Figure 3b

Figure 3a

Figure 4 shows the timing diagram of a TDL.



Figure

### Mapping the TDL

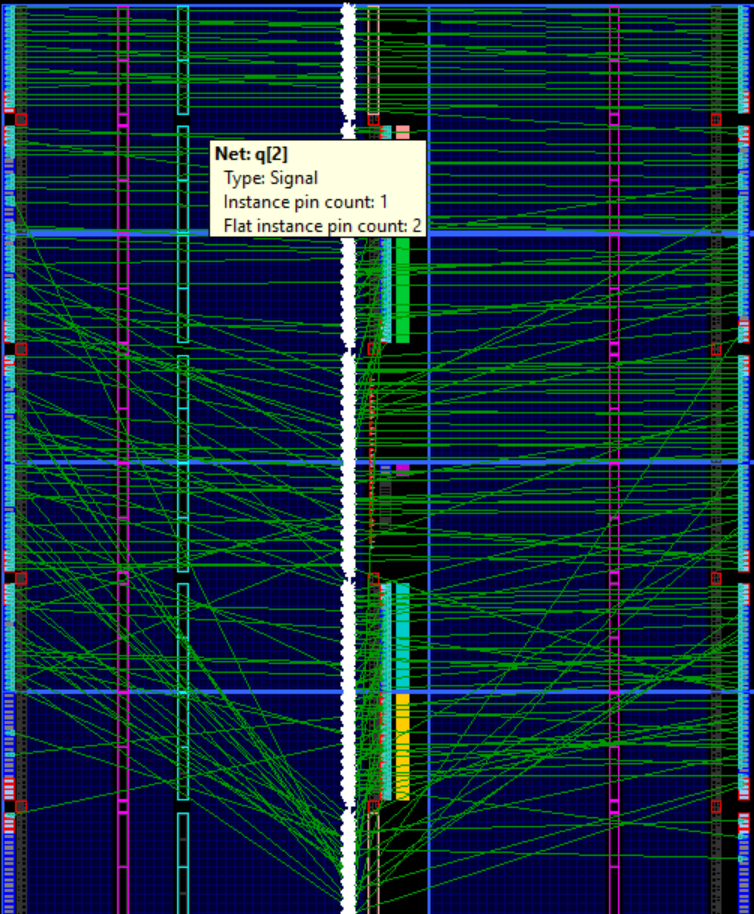
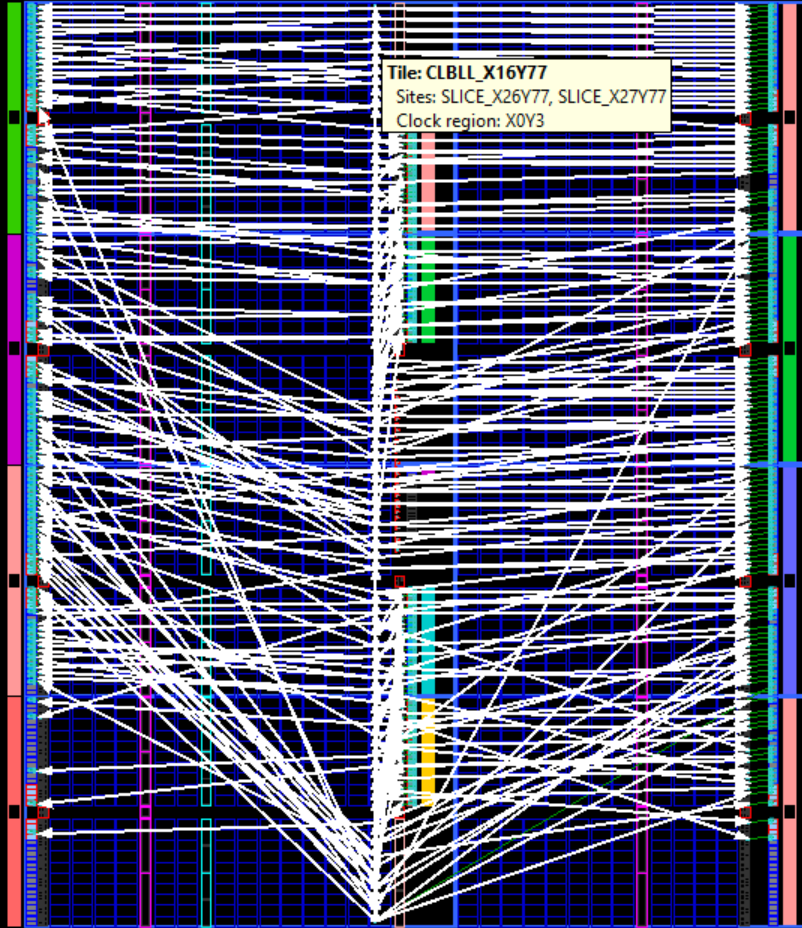
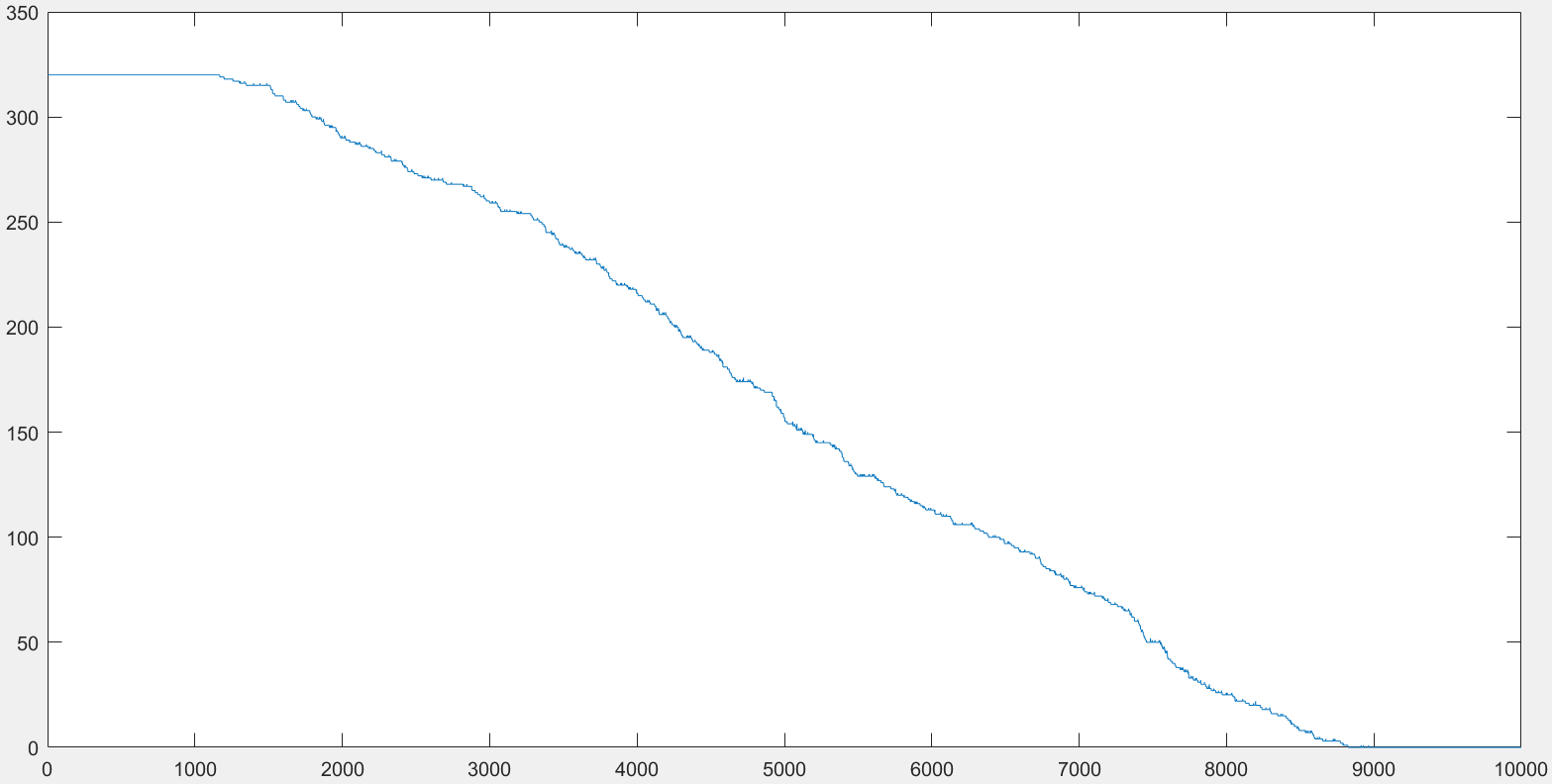
In Xilinx’s ISE 14.7, strategies and goals can be set for the project. These strategies and goals can affect how the TDL is mapped to the FPGA. Figure 5a shows how Xilinx maps the TDL with default settings. Figure 5b shows how it is mapped when the optimization goal is set to area and when registers are not mapped to IOB.

Figure 5b

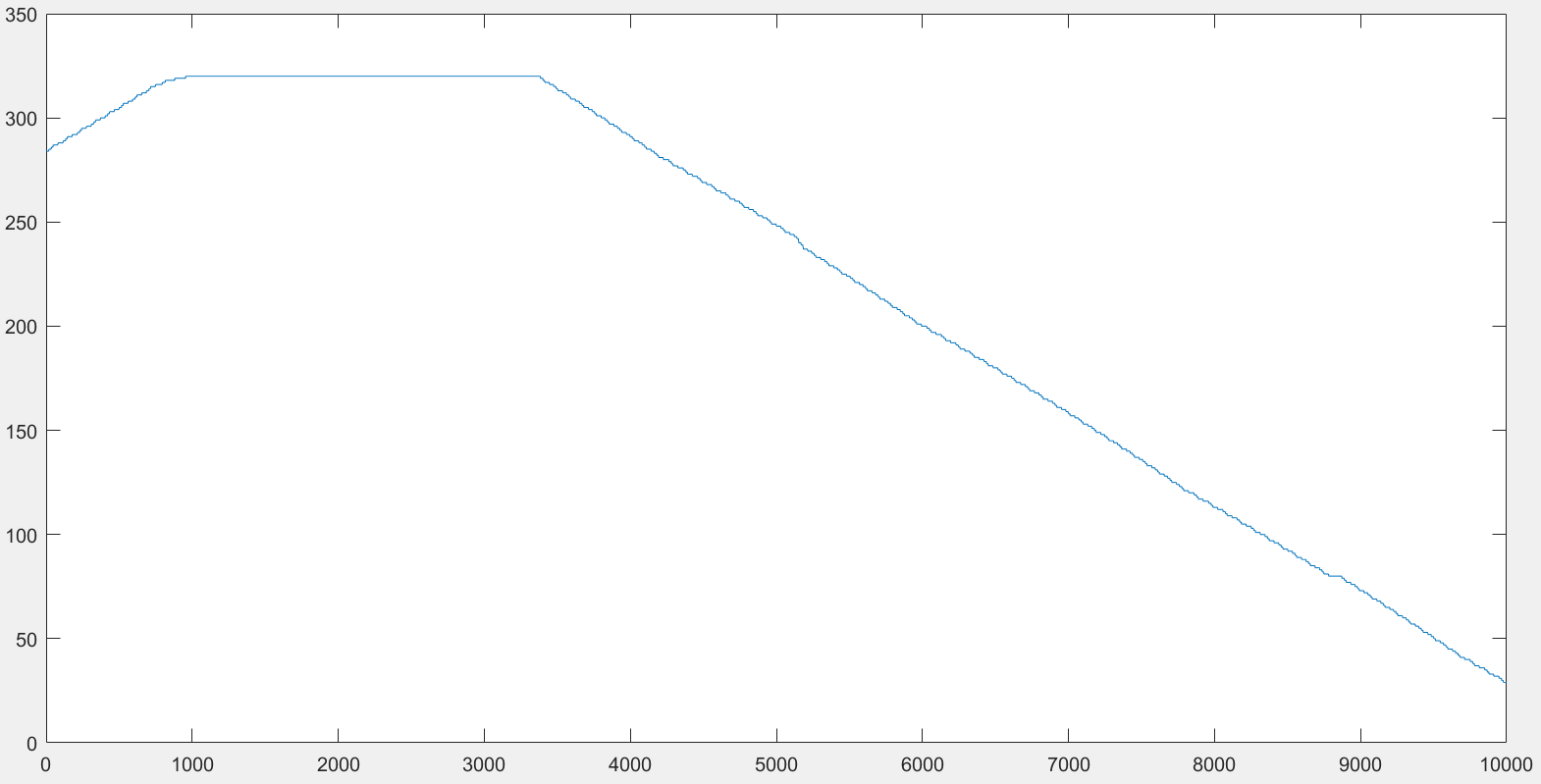
Figure 5a

The green lines represent connection to IO pins. In figure 5a the flip flops are put with the IO pins however, when area is the optimization goal and registers are not mapped to IOB, then the flip flops are put in the same slice as the CARRY4.

### Simulating TDL



Figure

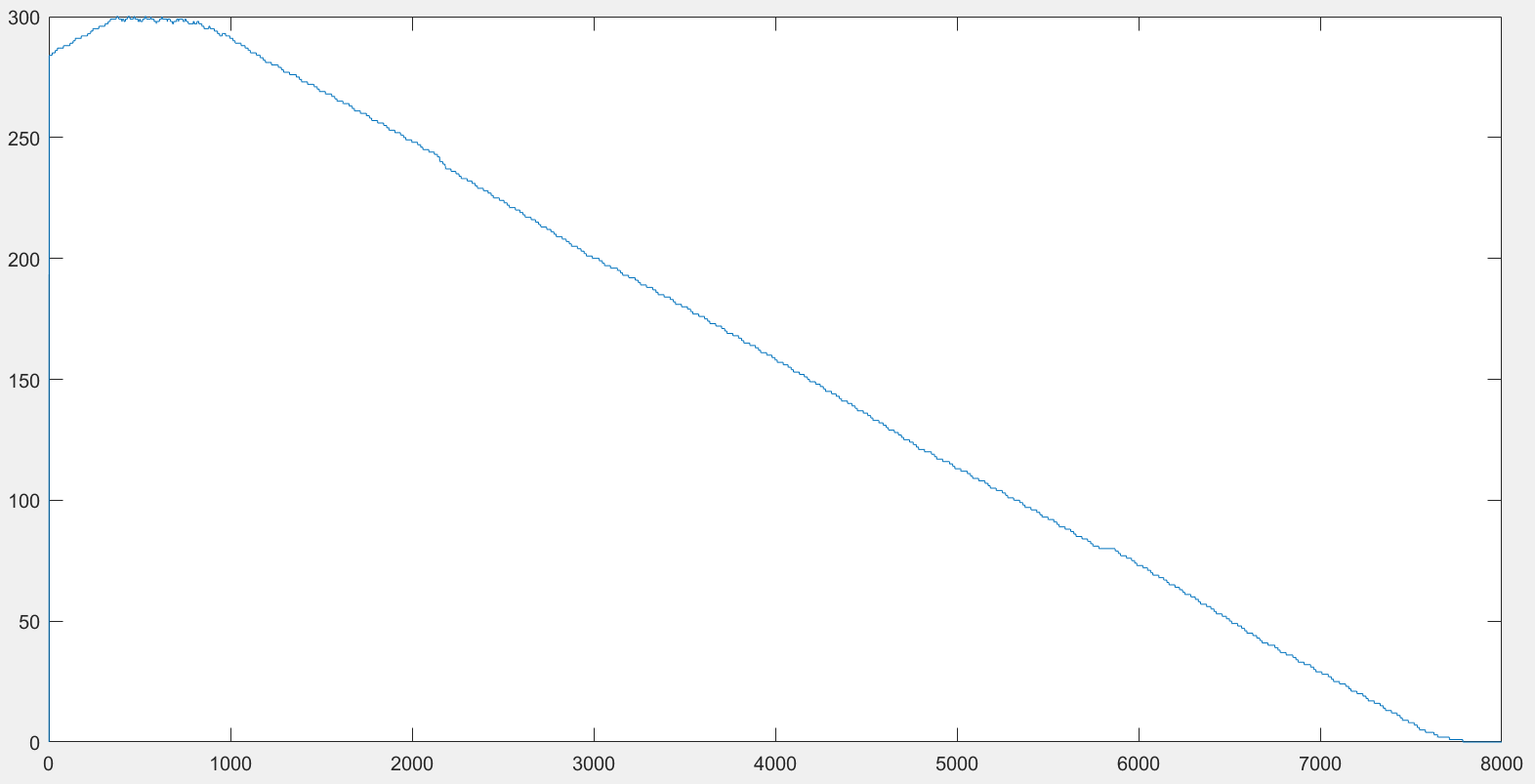


Figure

Figure 6 and figure 7 show data obtained from a TDL designed with eighty CARRY4 primitives and a ten-nanosecond clock. The x-axis represents the offset of the pulse. For example, at x = 0, the pulse came in 10 ns before the first rising edge, at x = 3,000, the pulse came in 7000 ns before the first rising edge, and at x = 10,000, the pulse came in 0 ns before the rising edge (came right with the rising edge). The y-axis represents the number of 1s that the TDL recorded for the given offset

Figure 6 shows the data produced with the default settings. We can see an initial plateau and then a steady decline. However, the line seems to be jagged and not as uniform as the slope in figure 7. The initial plateau can be attributed to the fact that it takes about 7.4 ns for the signal to propagate from the first CARRY4 instance to the last one. With a 10-ns clock, the TDL is too short to measure a full period. We can see that the declining portion of the graph can be estimated to be about 7,400 units long. This shows that our TDL can make measurements of pulses that come 7.4 ns before the rising edge, as stated before. Furthermore, we can note that the graph is shifted to the left. In a perfect scenario, with a 7.4 ns long TDL and a 10-ns clock we can expect a plateau for the first 3,000 units and then a steady decline to 0 at the 10,000th unit. However, this is not the case. The shift can be attributed to the fact that it takes some time for the trigger signal to reach the first CARRY4 instance.

Figure 7 shows the data produced when area is optimized. This graph is more shifted to the right and shows a more unusual behavior in the beginning. The initial rise in the graph is because there is a delay from when the rising edge to when q changes on the flip-flops. The pulse comes 10 ns before the clock edge and then goes low on the rising clock edge (the pulse is 10 ns long). In an ideal situation, all the flip-flop’s q outputs should show 1. However, in this case since there is a delay before q changes, then there is some time for the first CARRY4 instances to start changing to 0. As the 10-ns pulse is shifted to past the clock edge we see that less of the CARRY4 instances are given the time to change to 0 and the plateau is approached. The plateau can be explained by the shortness of the TDL, as stated previously. In figure 7 we see that the shift is so great that graph never reaches zero. We can say that when Xilinx is given the liberty to map the flip-flops, it will map them in a way to minimize the delay from rising edge to q. However, this increased the delay from CARRY4 to the d input of the flip-flop and this caused the jaggedness of the second graph. We chose that the second graph is better and decided to rerun the simulation with a faster clock. Specifically, a 7-ns clock for 8,000 iterations. 1,000 interactions more than needed.



Figure

Figure 8 shows a more promising graph that is uniform and has a full spectrum of 7 ns. However, we still see a plateau and rising graph in the beginning. This is because the pulse is exactly 7 ns. What is happening is that the pulse is starting to exit the TDL; the first CARRY4 instances are turning to 0. The plateau in the beginning is happening because the pulse is 7 ns long and the TDL is longer than 7 ns. pulse is simply appearing at various places in the TDL. Like so:

00001111…11110000

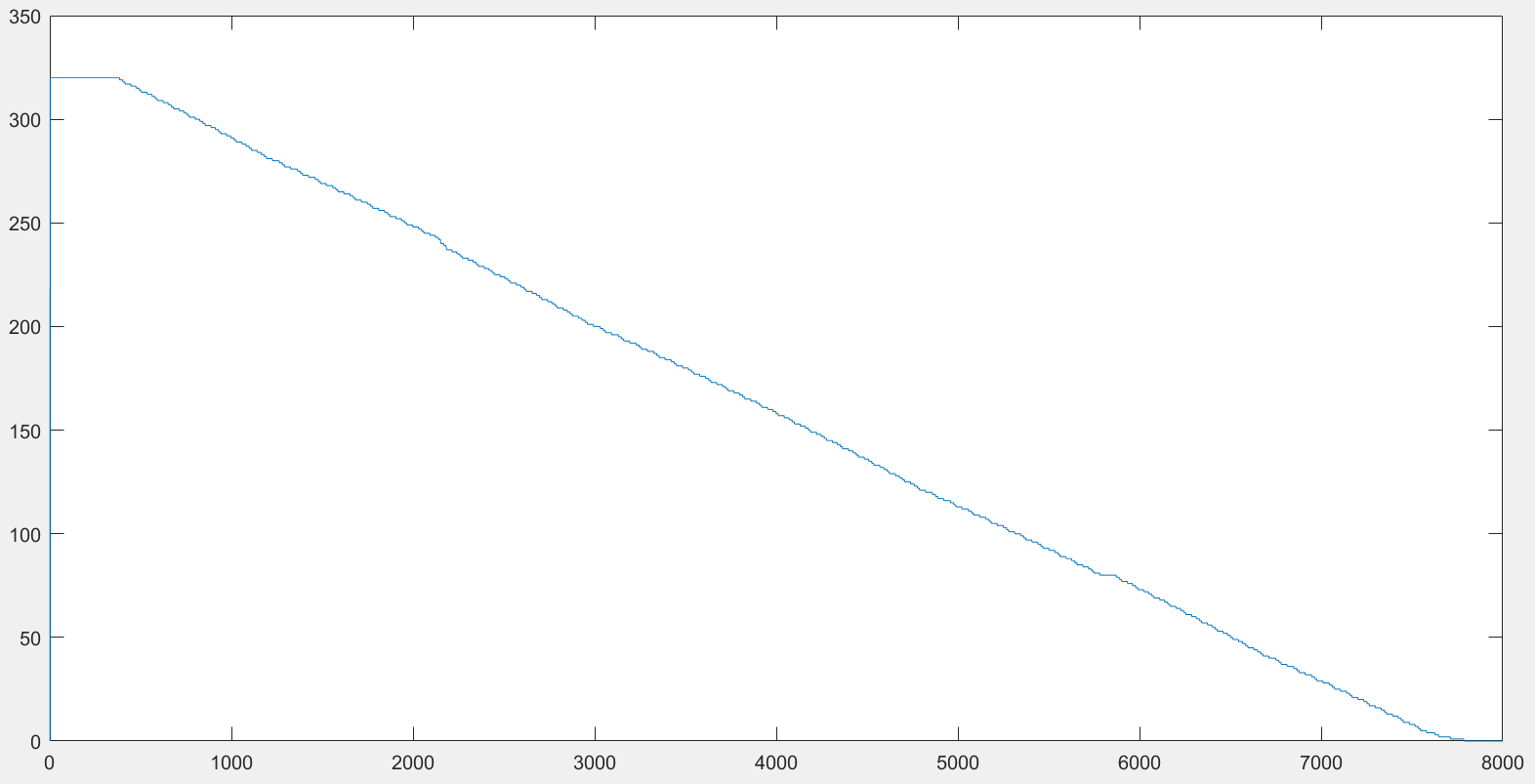
0001111…111100000

001111…1111000000

01111…11110000000

1111…111100000000

A solution should be to make the pulse longer and the positive sloped portion of the graph should disappear.



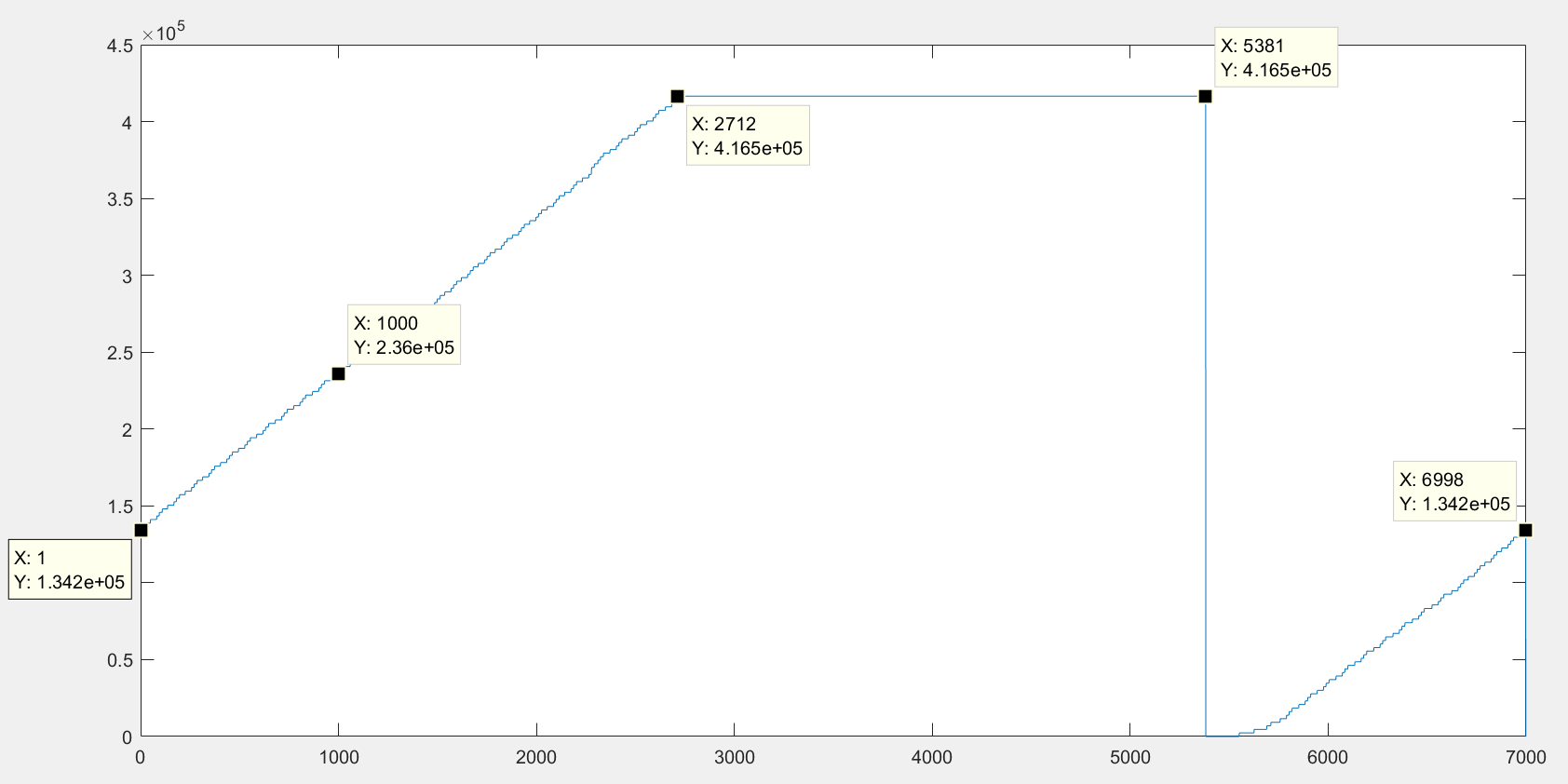
Figure

Figure 9 shows the relationship of high bits in the output of the TDL and how far before the rising edge the signal appears. We no longer see a positive slope in the beginning and only a plateau. This is explained by the shift caused by the delay of the flip-flops being triggered. We can imagine that the plateau in the beginning is when the pulse comes more than 7.4 ns before the rising edge which we are observing. The plateau at the end indicates that the trigger signal comes after the rising edge. The flip-flops are seeing a shifted clock and therefore we get a shifted graph.



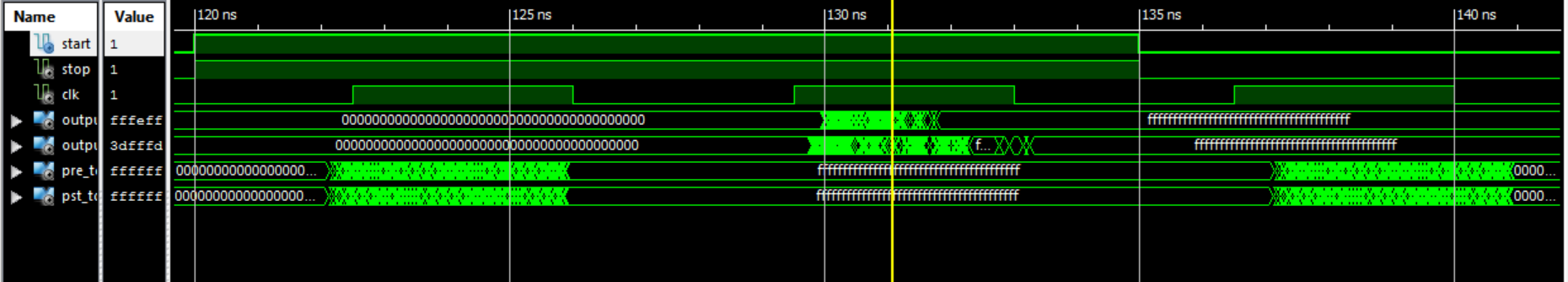
Figure

Shows a simulation waveform that shows that the TDL is about 7.4 ns longs.



## Two TDLs Mapped

When running a simulation with two TDLs, with the *‘start’* and *‘stop’* signals each going into one, we notice that both produce different results. This is an issure because if they produce very different results then we must characterized each one seperatley and account for different delays. This would complicate the math to get a result and would also add greater error to our result. To solve this issue we use the Plan Ahead tool by Xilinx to map the TDLs and *‘start’* and *‘stop’* signals to specific parts of the FPGA so that the two TDLs notice their respective input at the same time. We found such a placement that the difference in time form when the least-significant bit in each TDL toggled to ‘1’ is 14 picoseconds.



How much time it takes for the LS-CARRY4 to see the input signal.

Pre\_tdl: 2.138 ns

Pst\_tdl: 2.124 ns

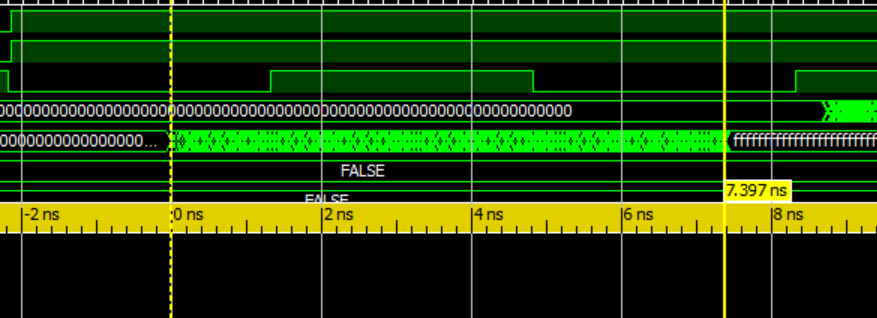
7.3 ns the flip flops start to change on the pst\_tdl

7.4 ns || || || ||pre\_tdl

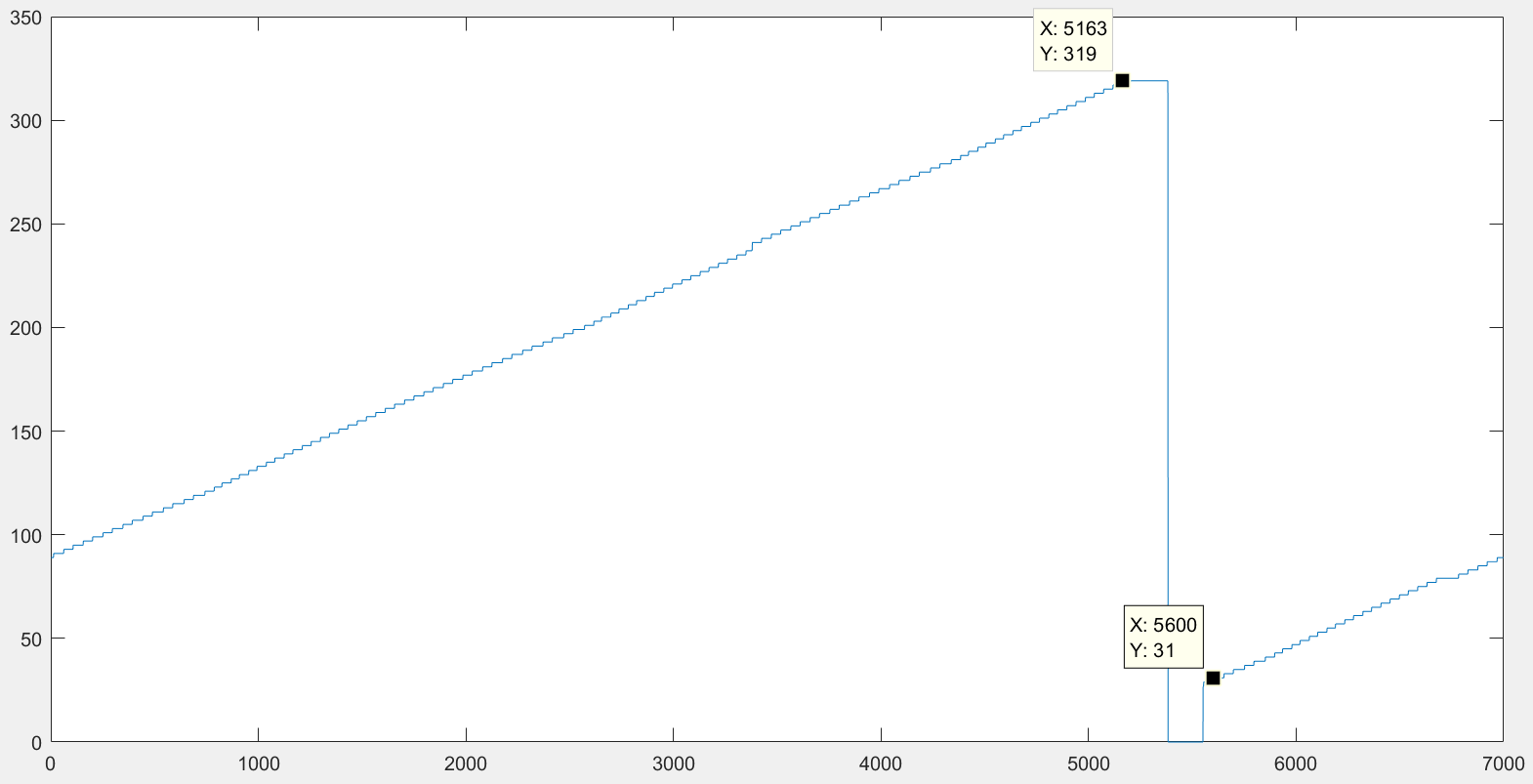
After removing pst\_tdl and extending pre\_tdl to 80 bits and keeping all constraints the pre\_tdl still shows the same characteristics. Which is good.

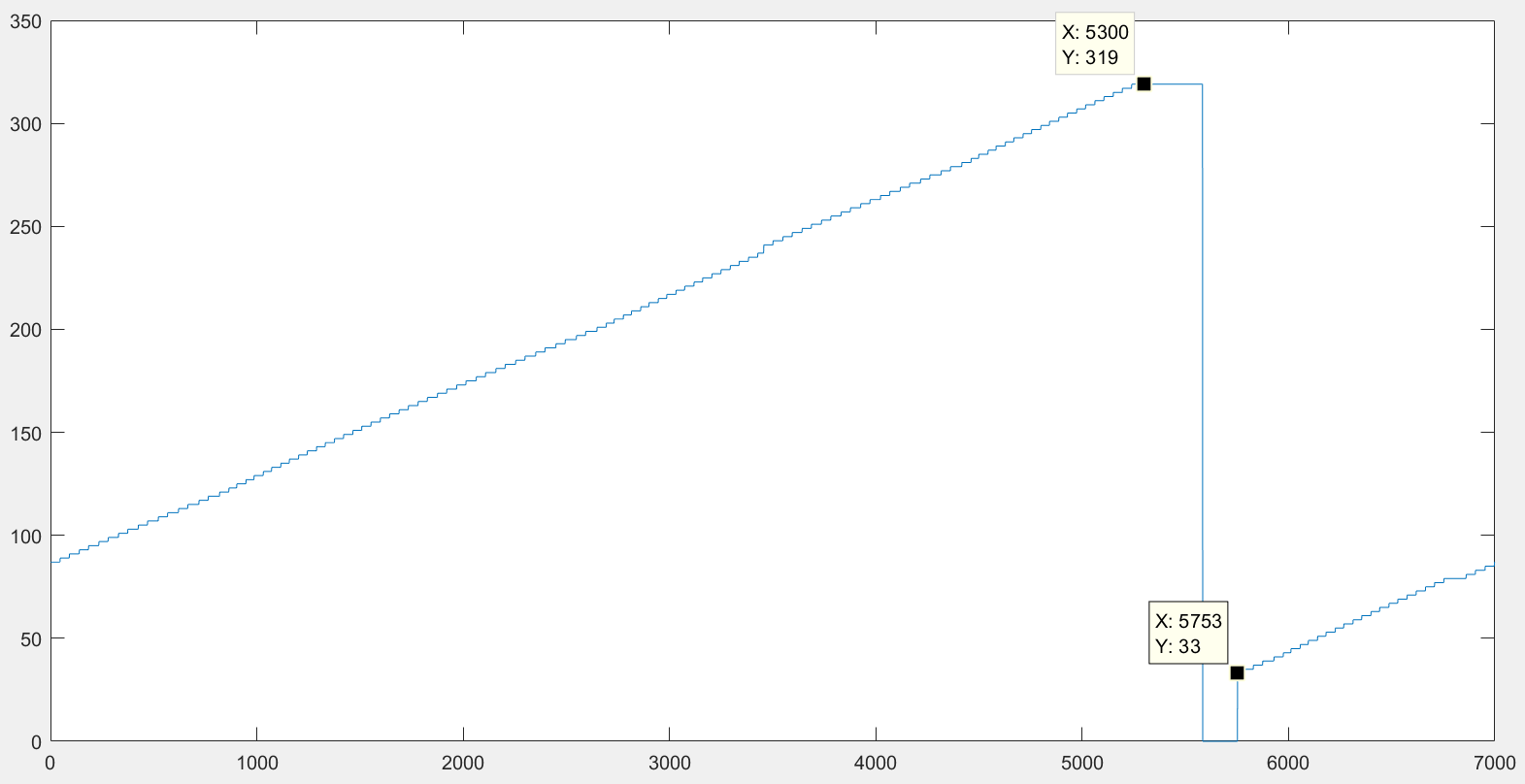
Start saving data at 10.5 ns after rising edge of interest

115.5 ns the signal should start.

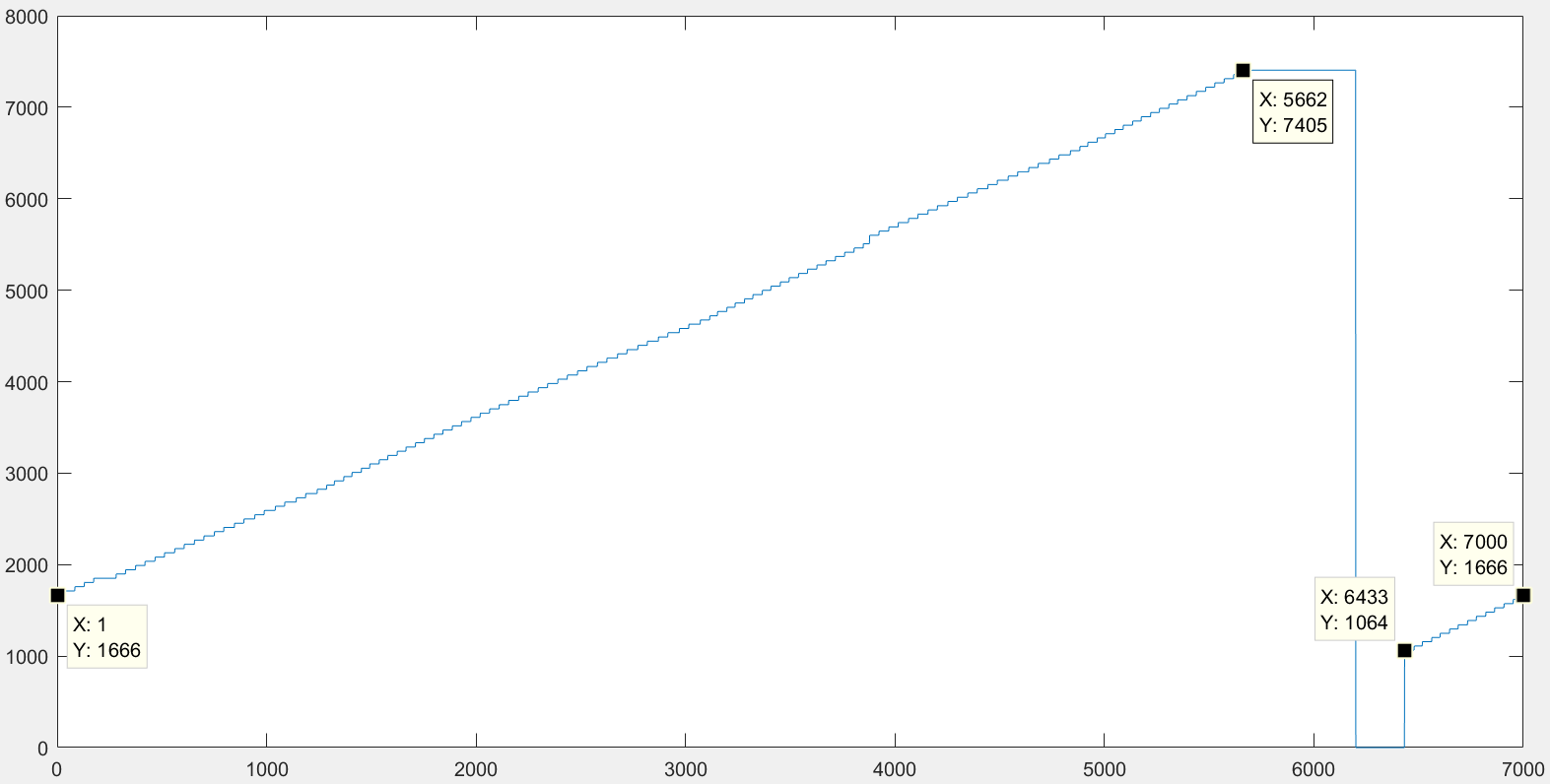


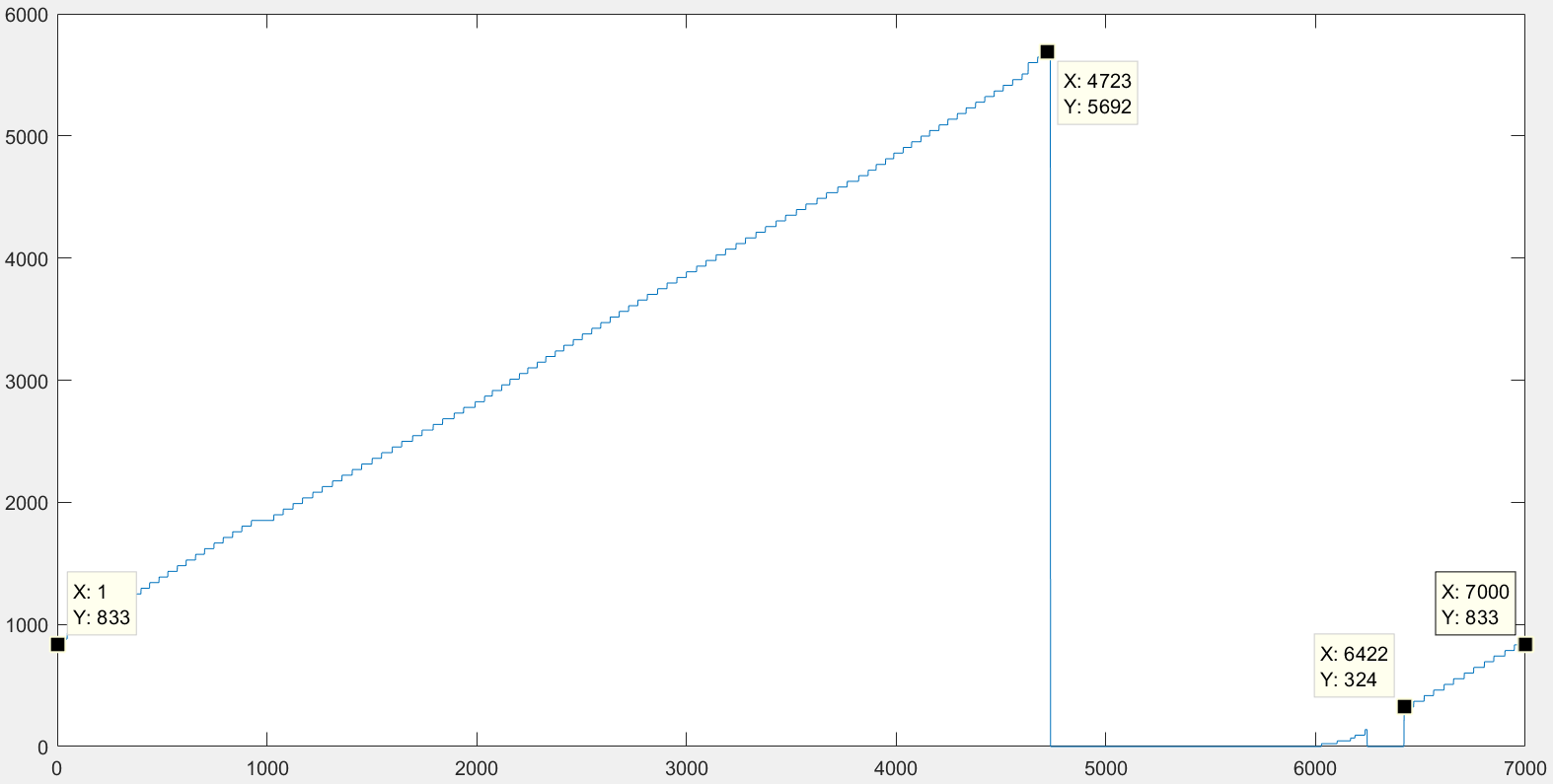
Pre\_tdl length is shown to be 7.397 ns in this scenario.

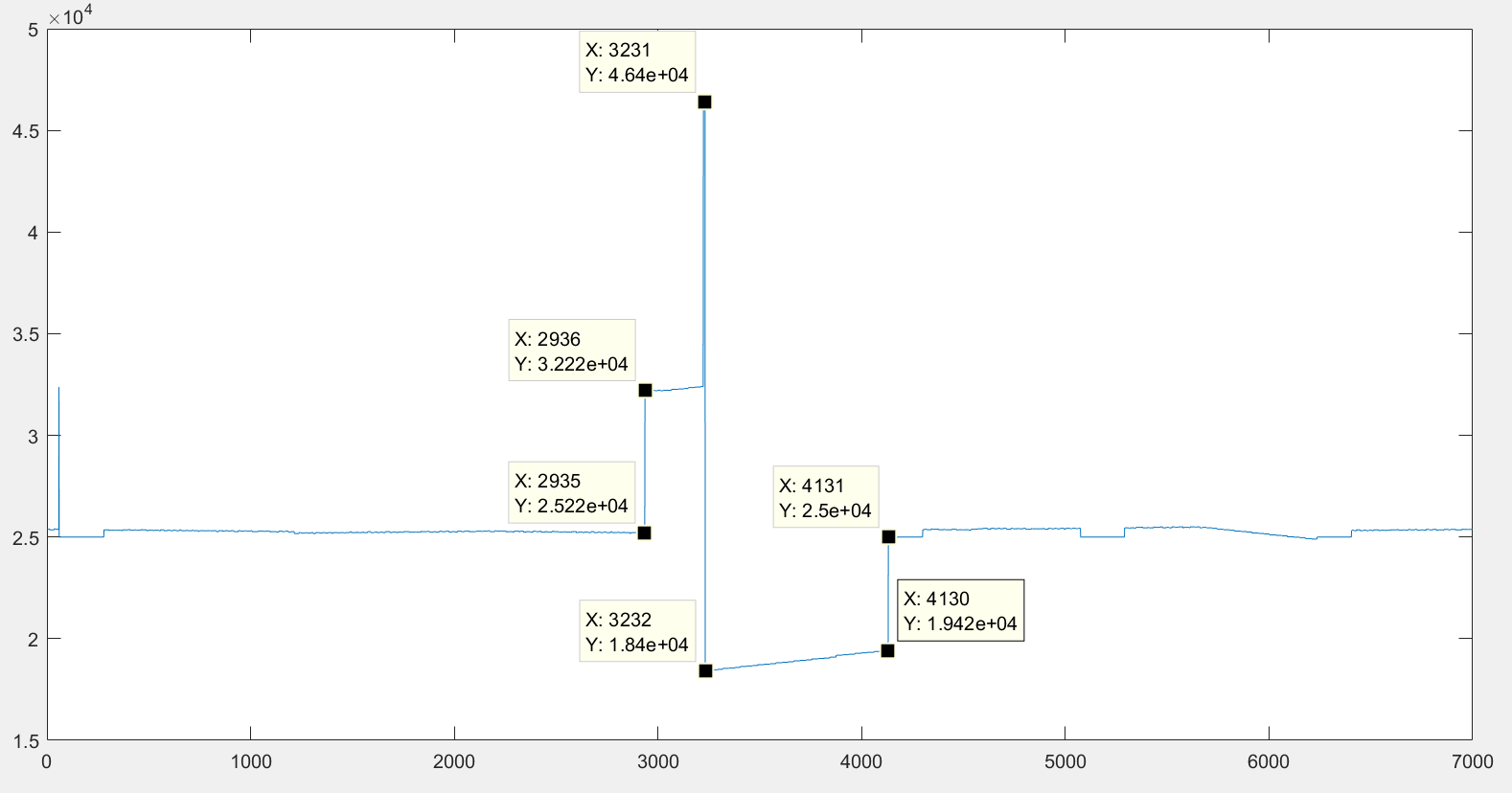




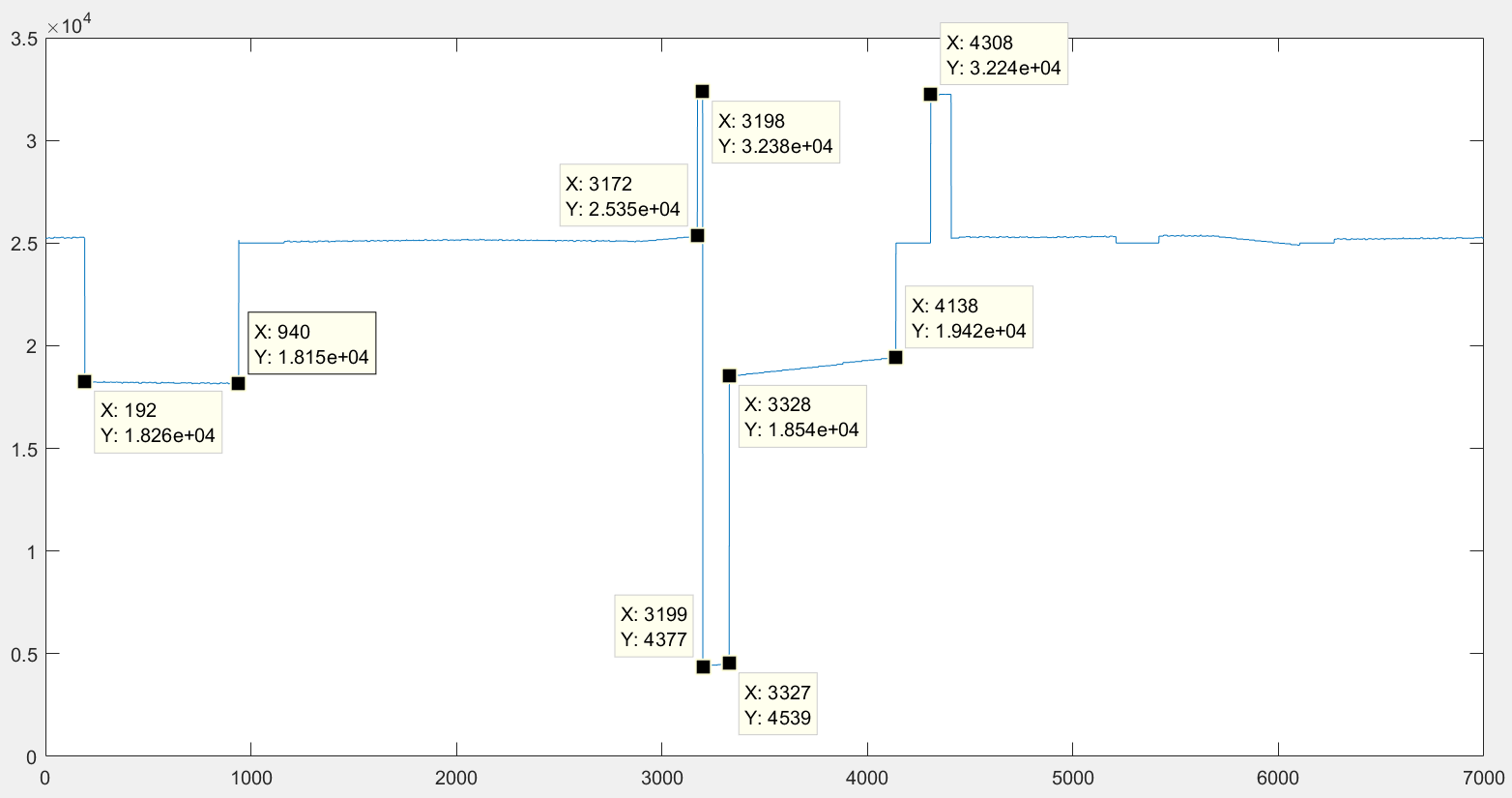
For the humming weight counter. When it is nonsynchronous for measuring 320 bits then it takes bout 50 ns.

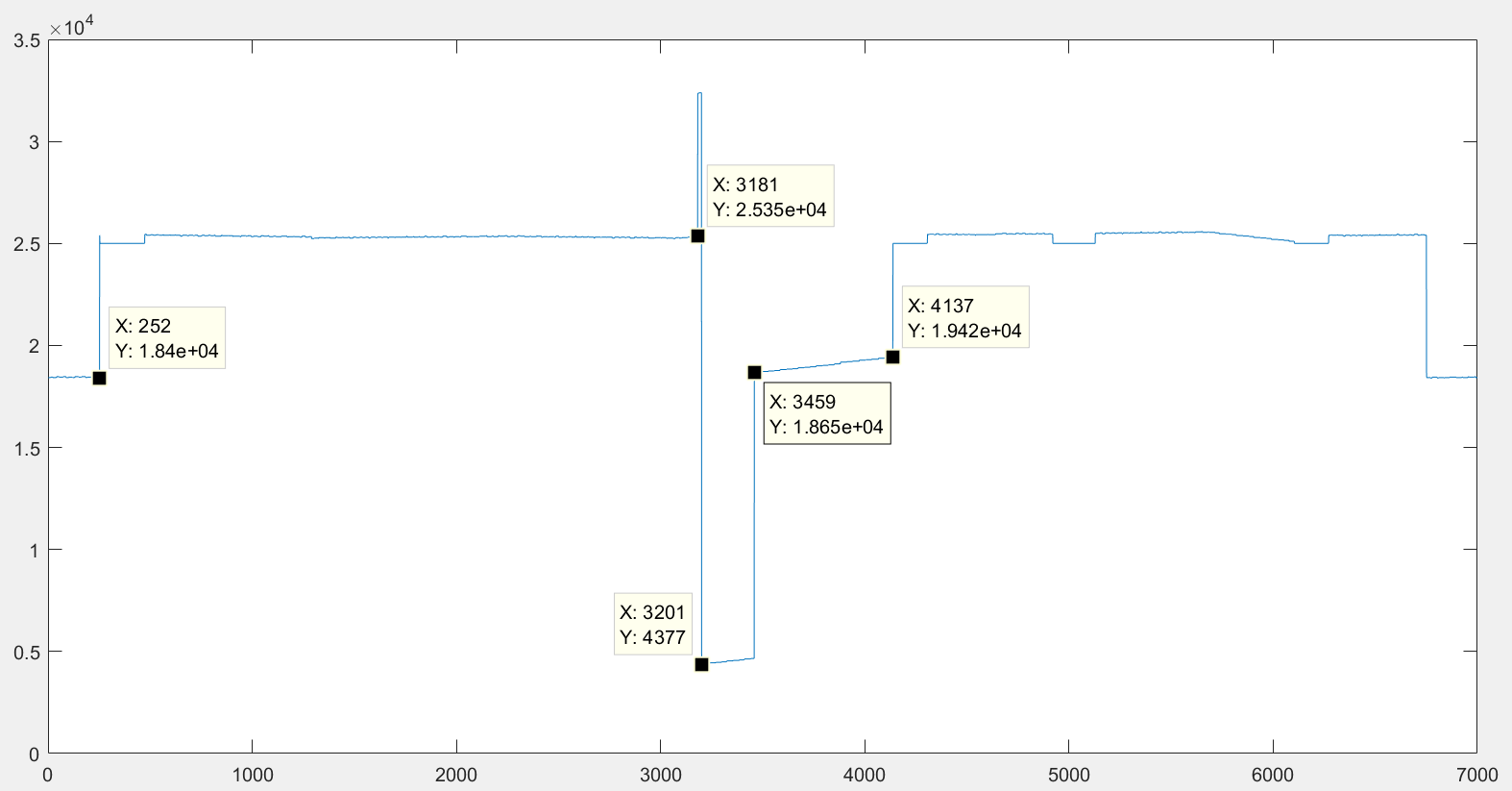


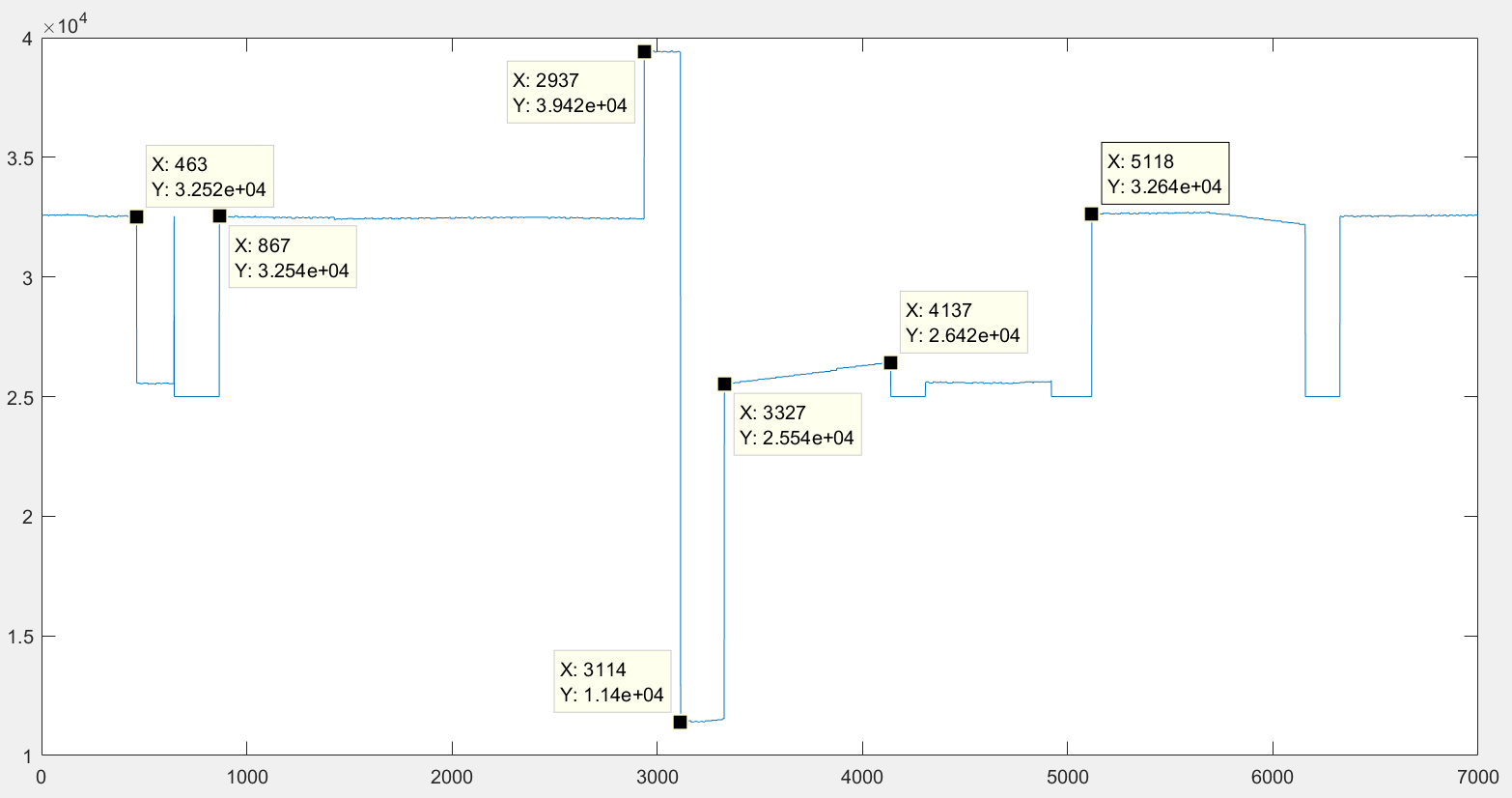


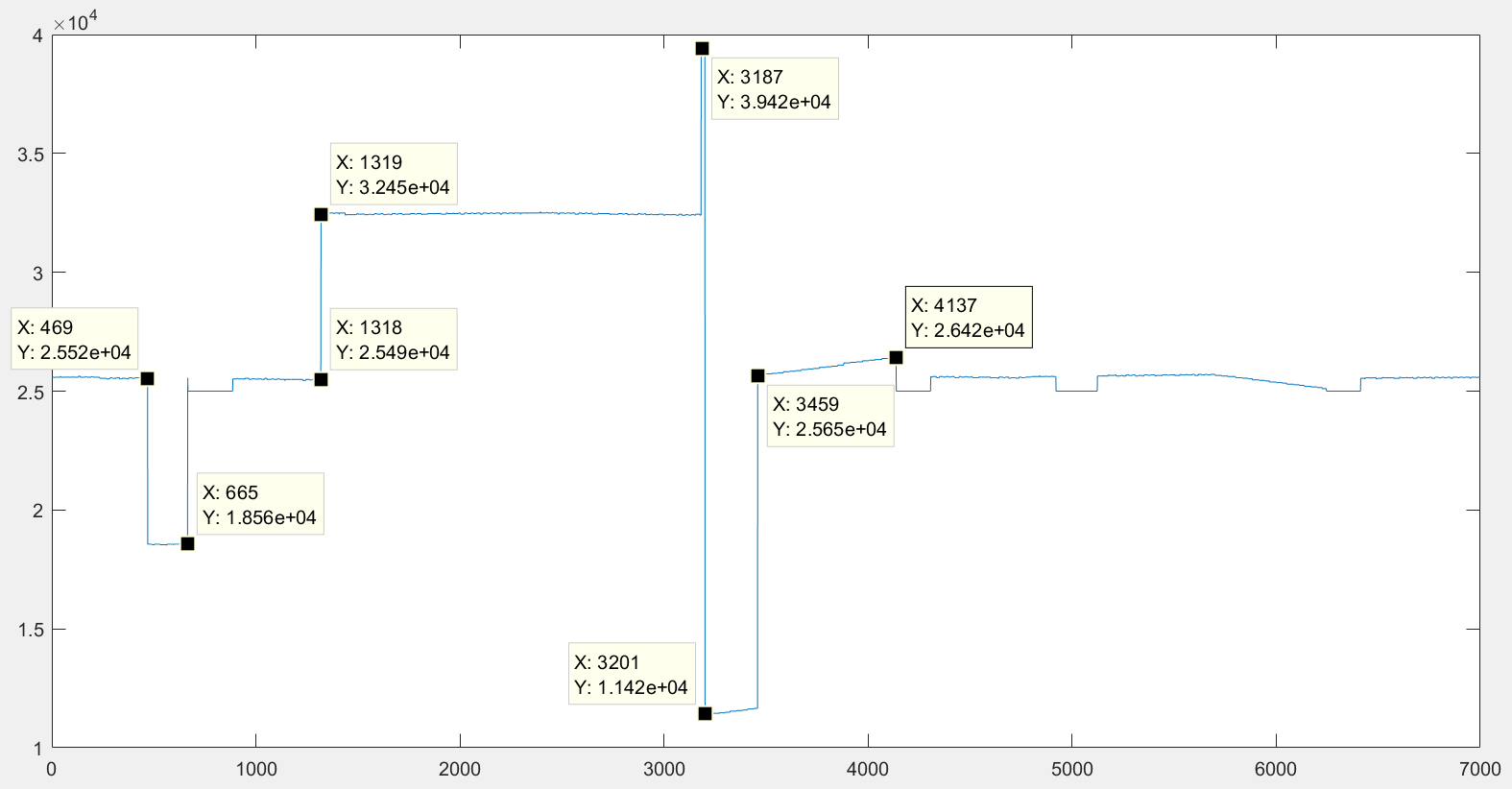


Pre\_bit\_cnt > 231



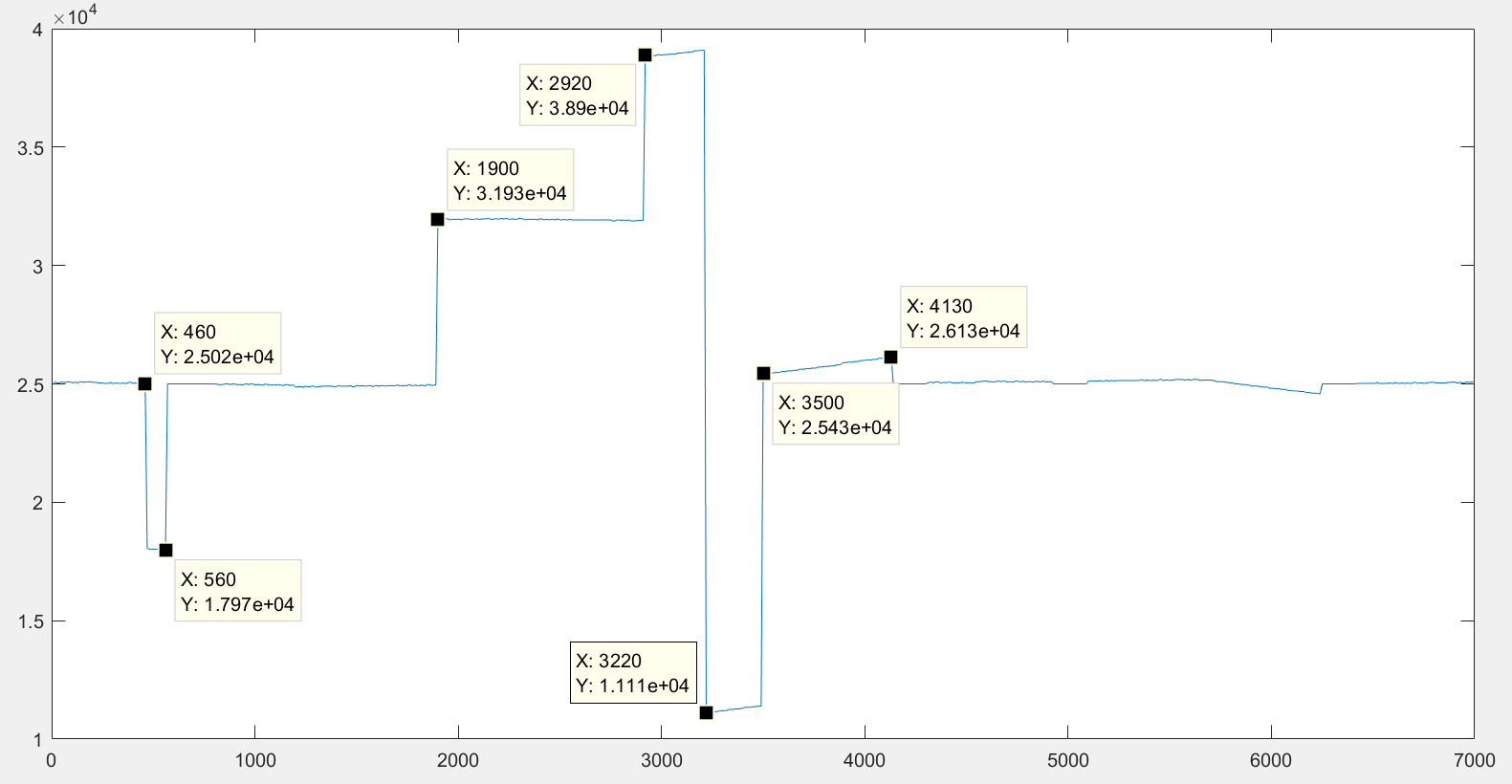






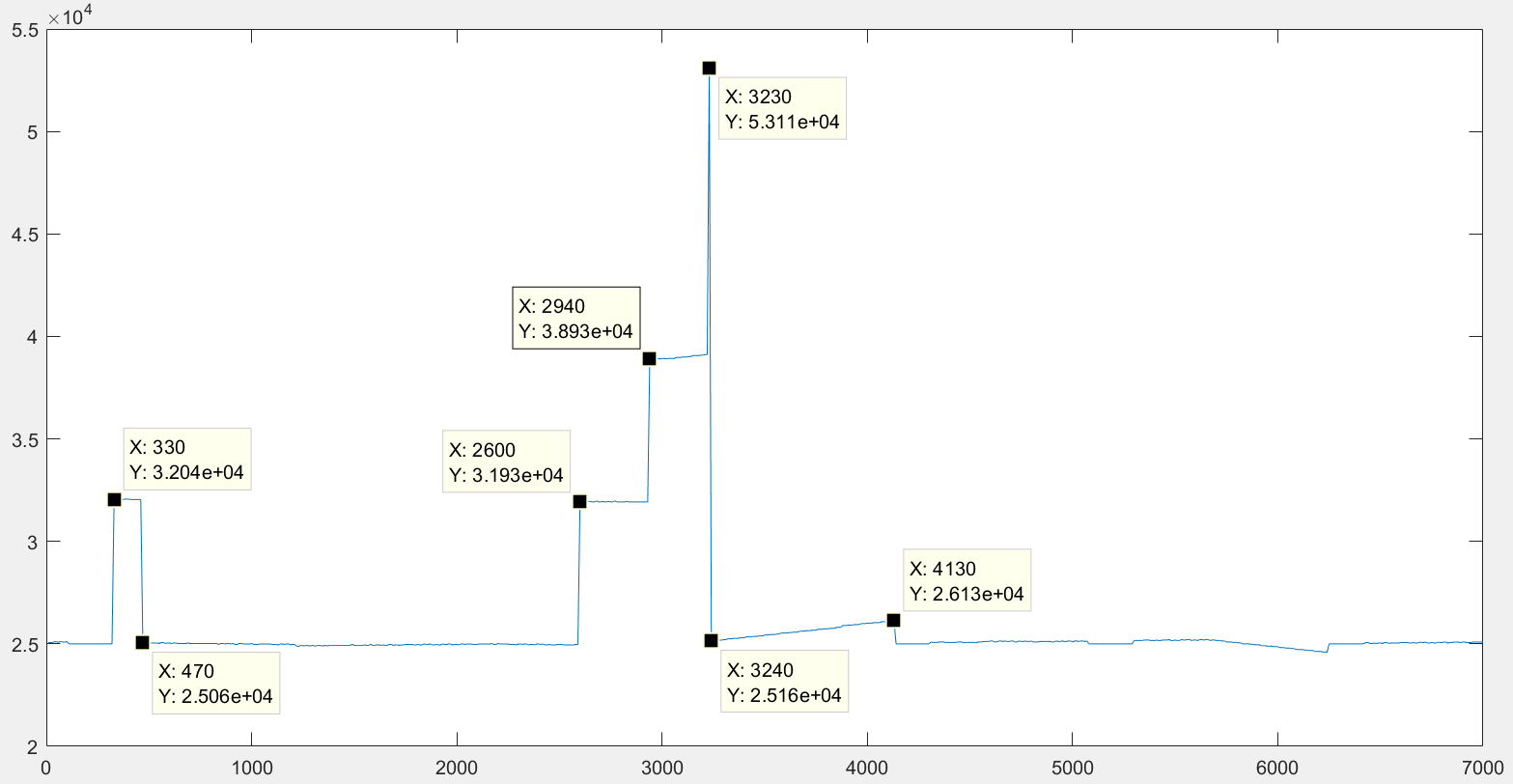
Pre\_bit\_cnt > 231

Pst\_bit\_cnt < 87



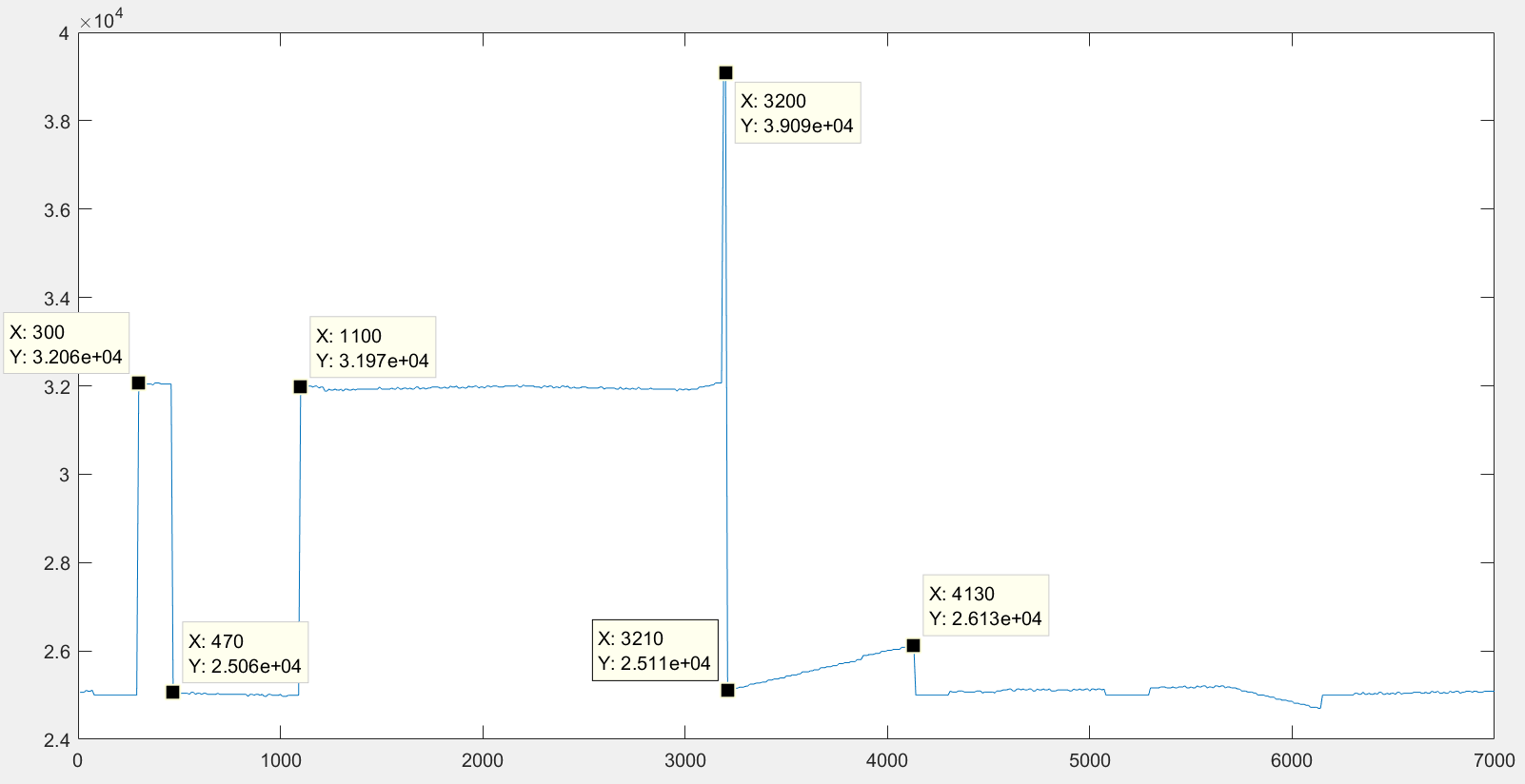
Pre\_bit\_cnt > 231

Pst\_bit\_cnt < 50



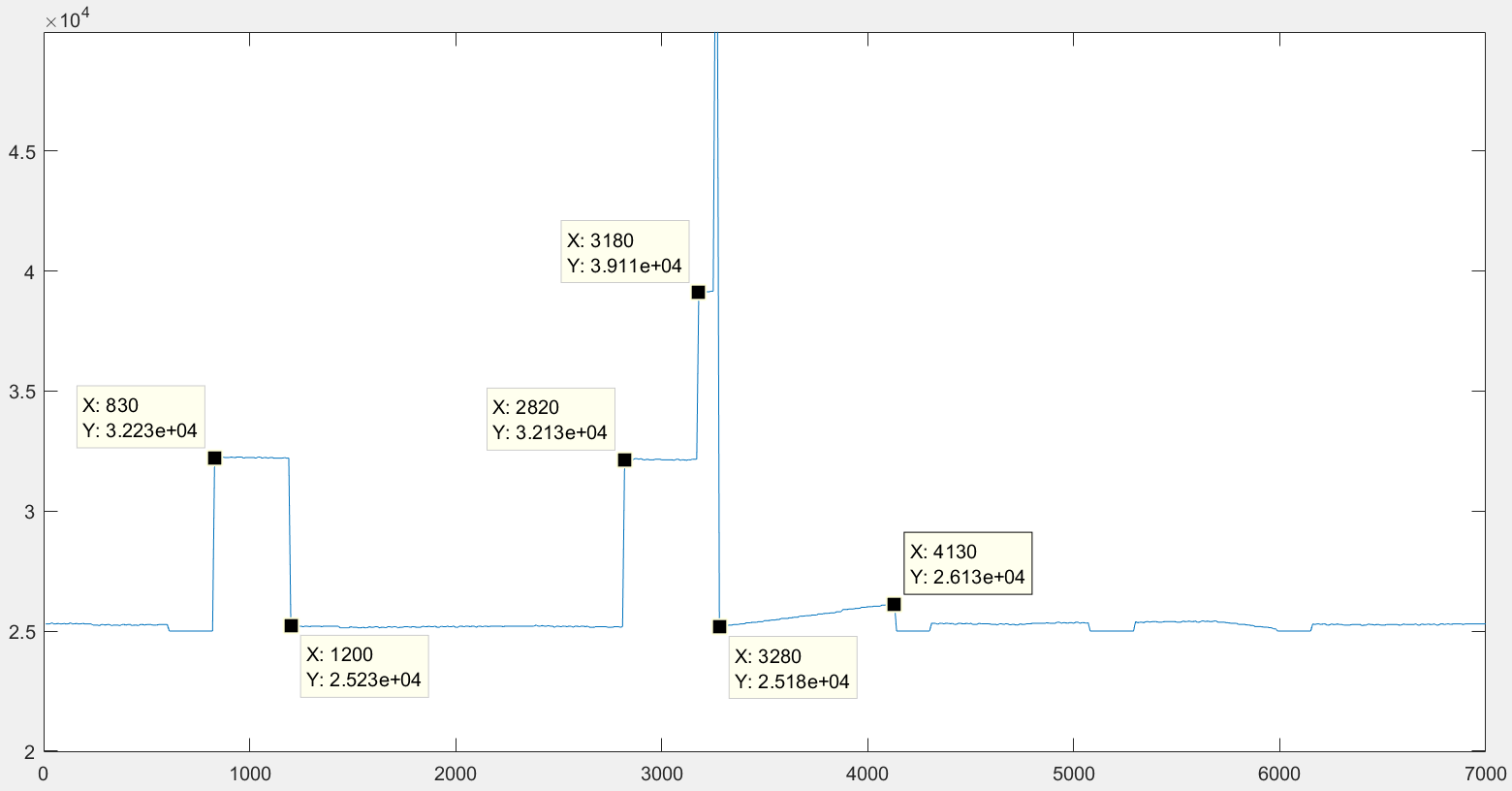
Pre\_bit\_cnt > 231

Pst\_bit\_cnt < 20 no nested if statements



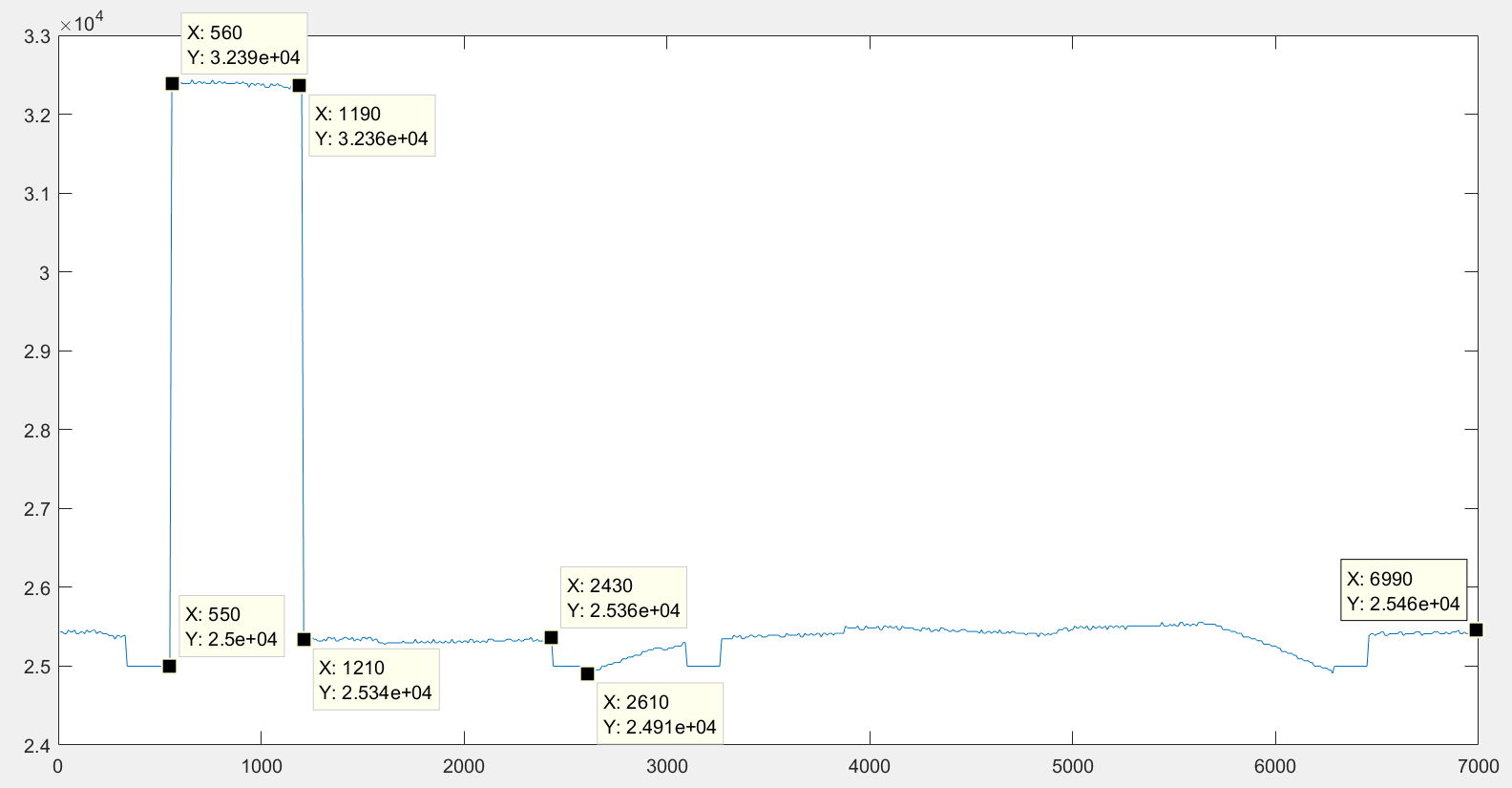
Pre\_bit\_cnt > 231

Pst\_bit\_cnt < 87 no nested if statements



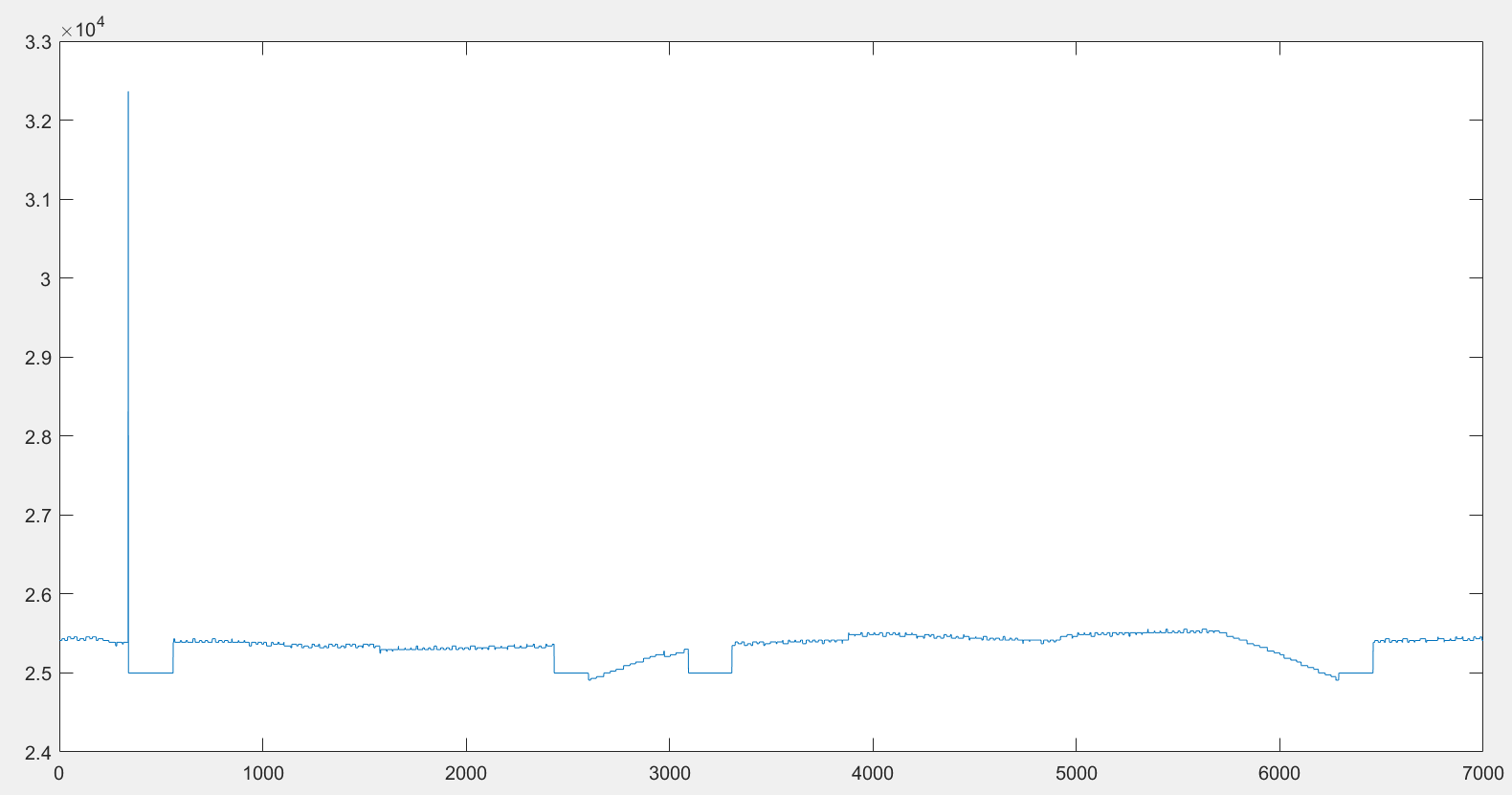
Pre\_bit\_cnt > 200

Pst\_bit\_cnt < 20 no nested if statements



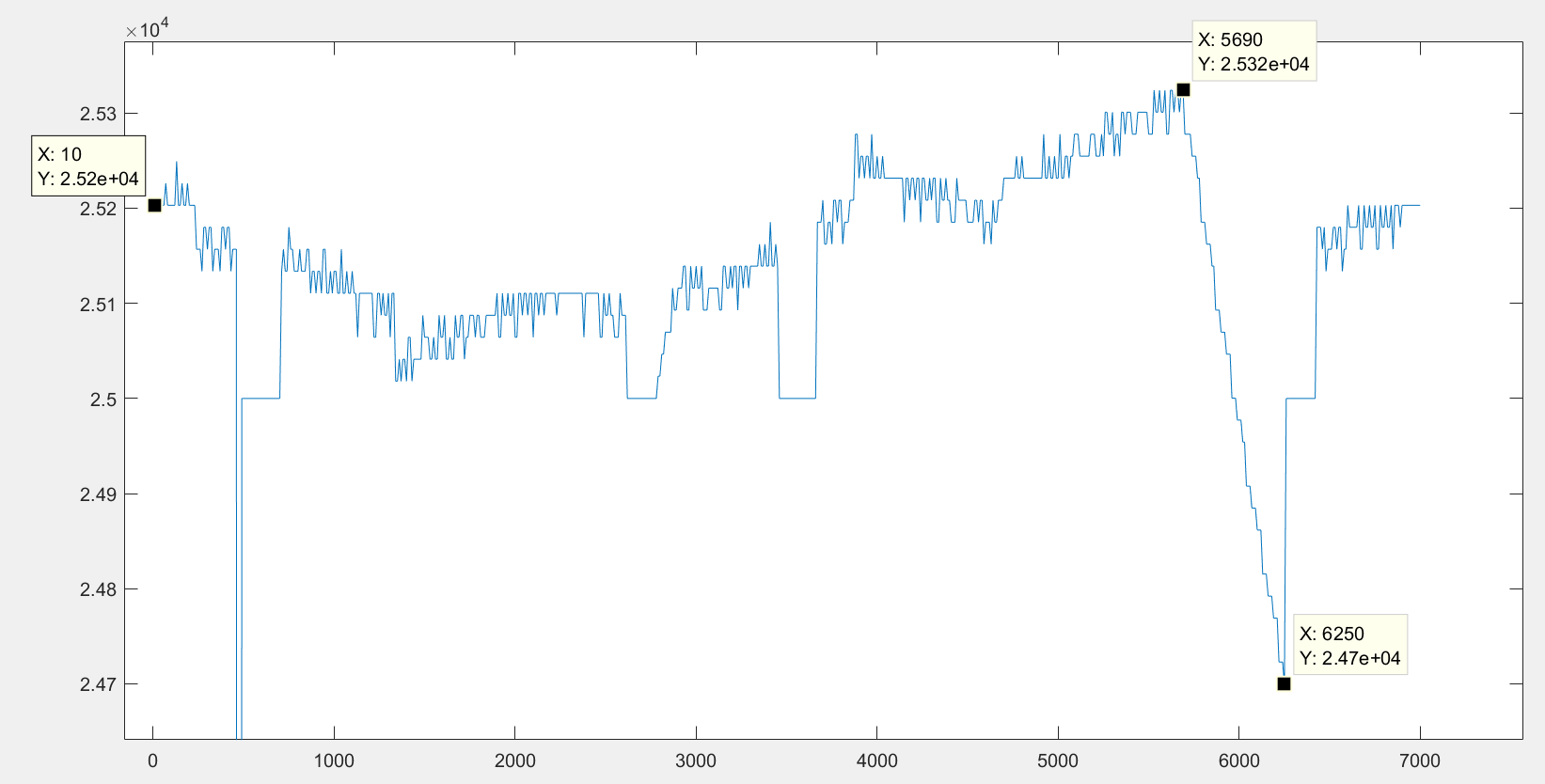
Pre\_bit\_cnt > 200

Count was saved on the pst\_trig.



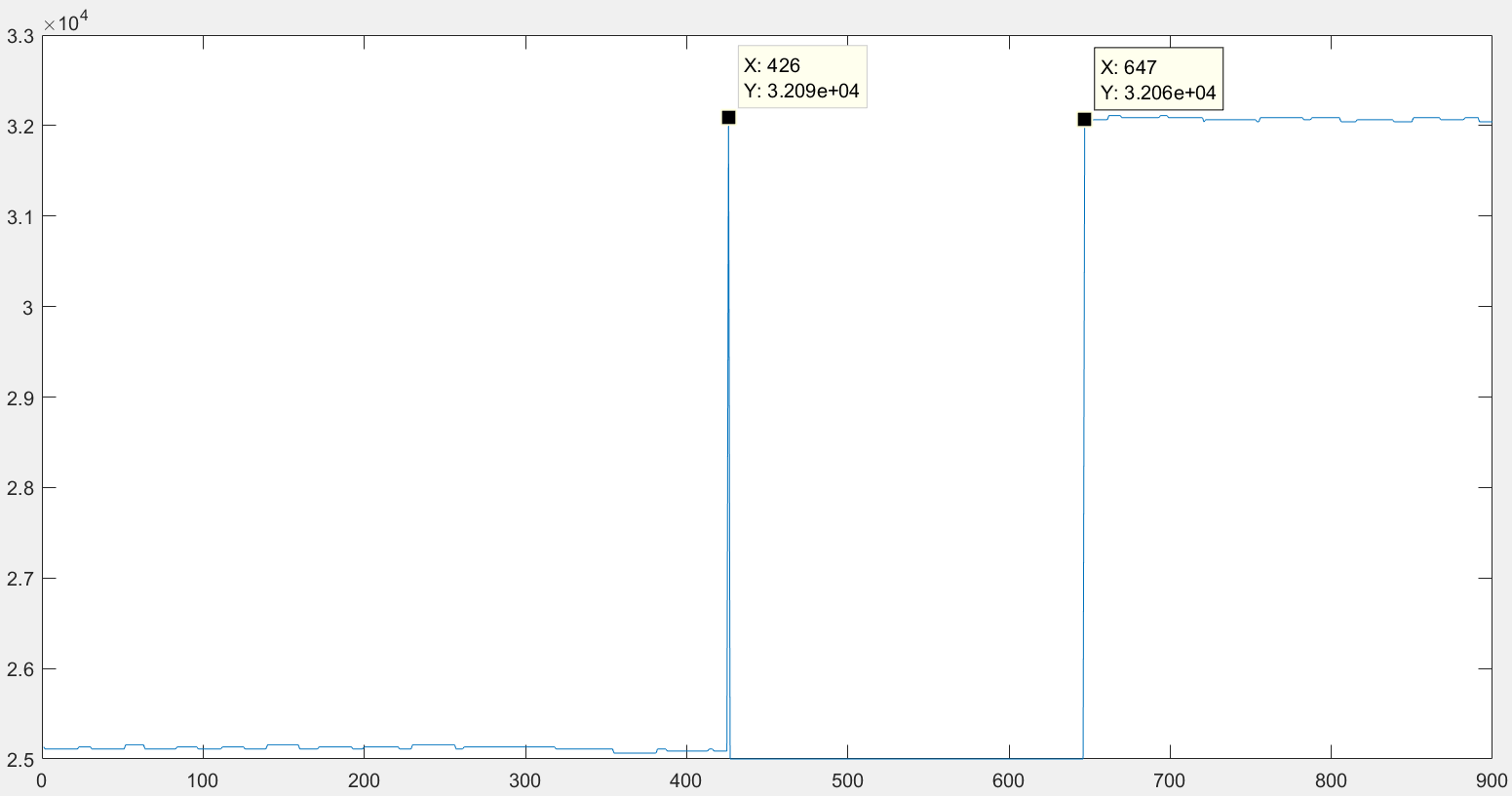
Pre\_bit\_cnt > 228

Count was saved on the pst\_trig.



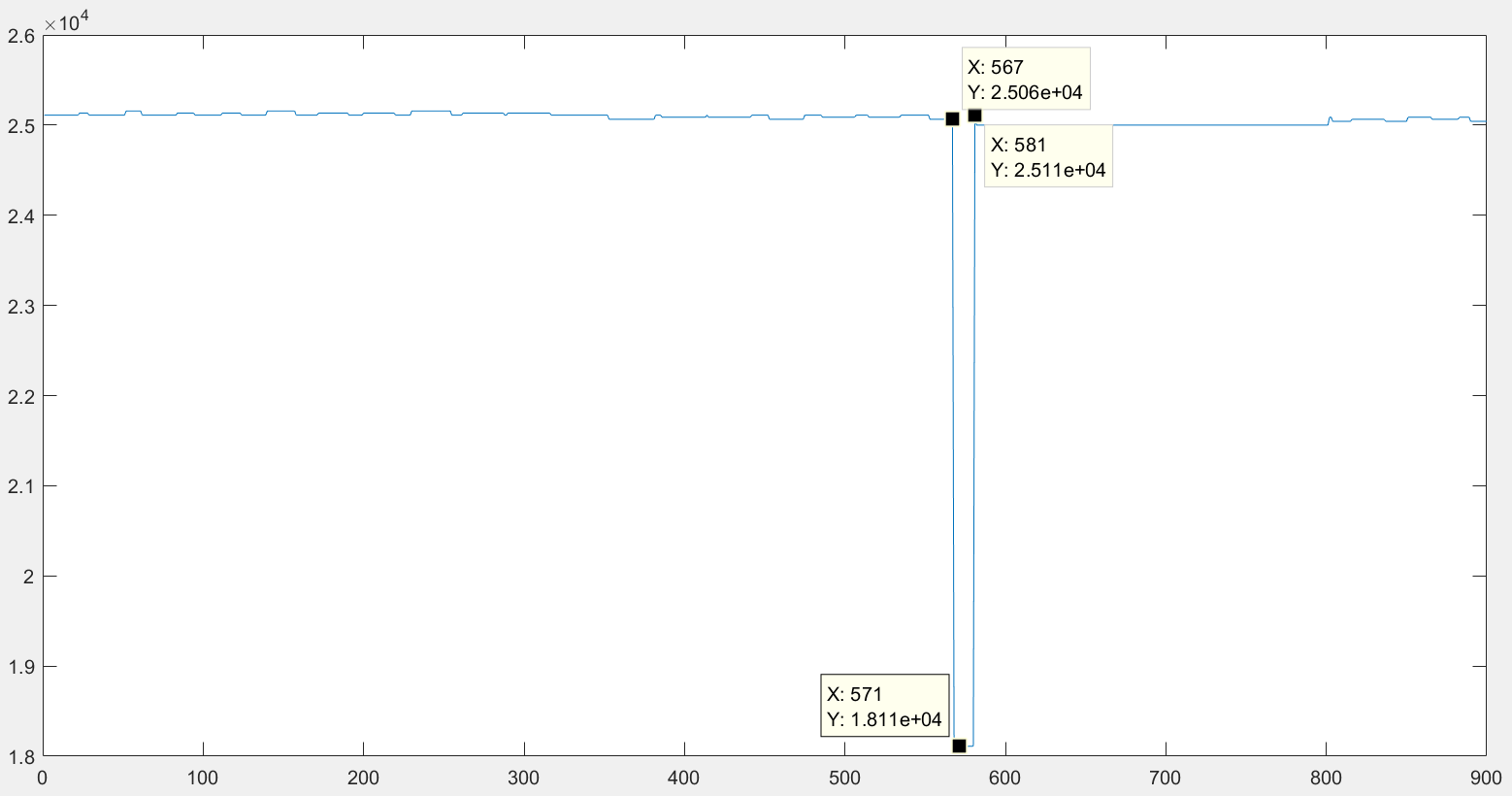
Pre\_bit\_cnt > 231

Count was saved on the pst\_trig.



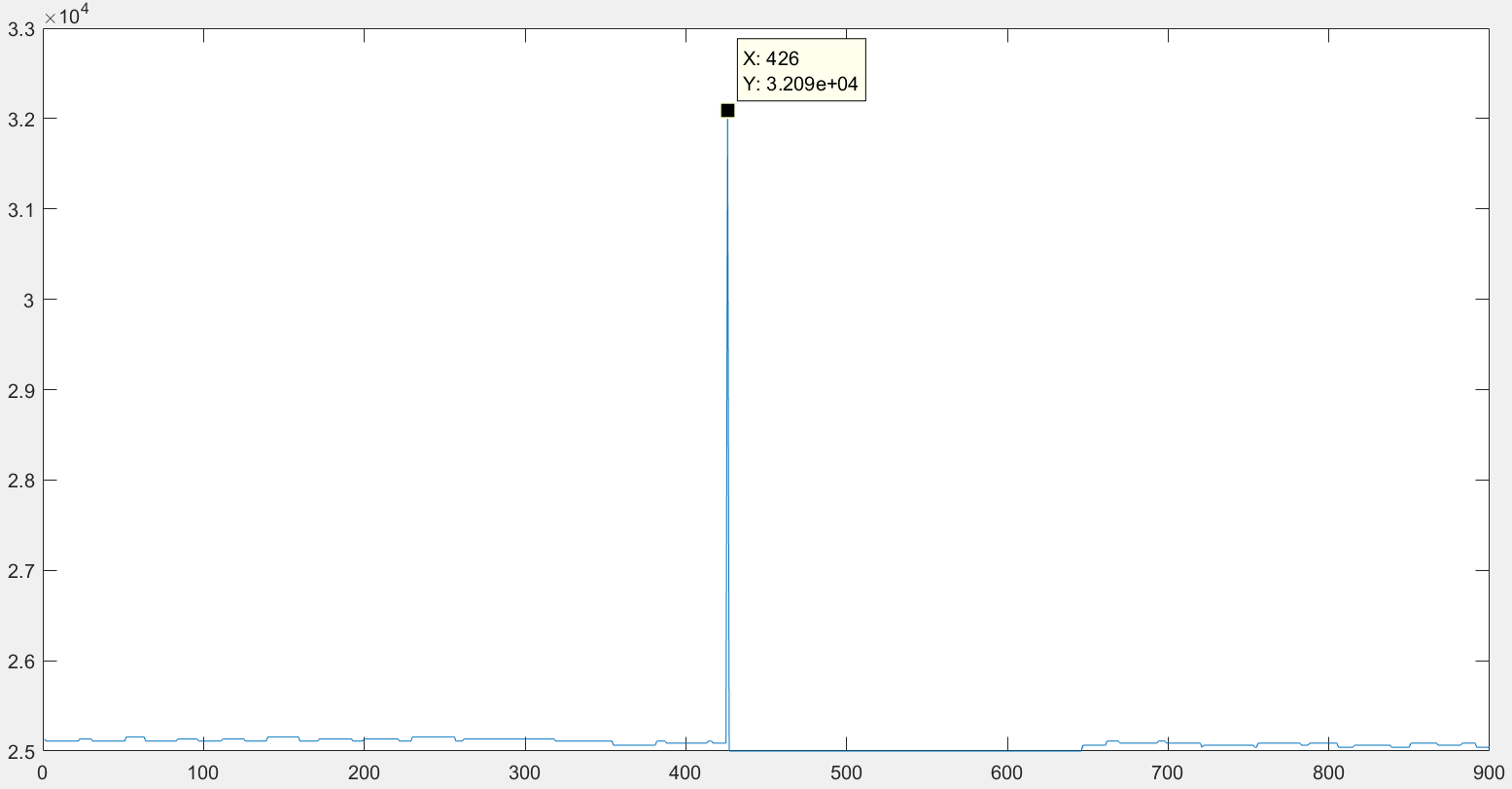
Pre\_bit\_cnt > 220

Count was saved on the pst\_trig.



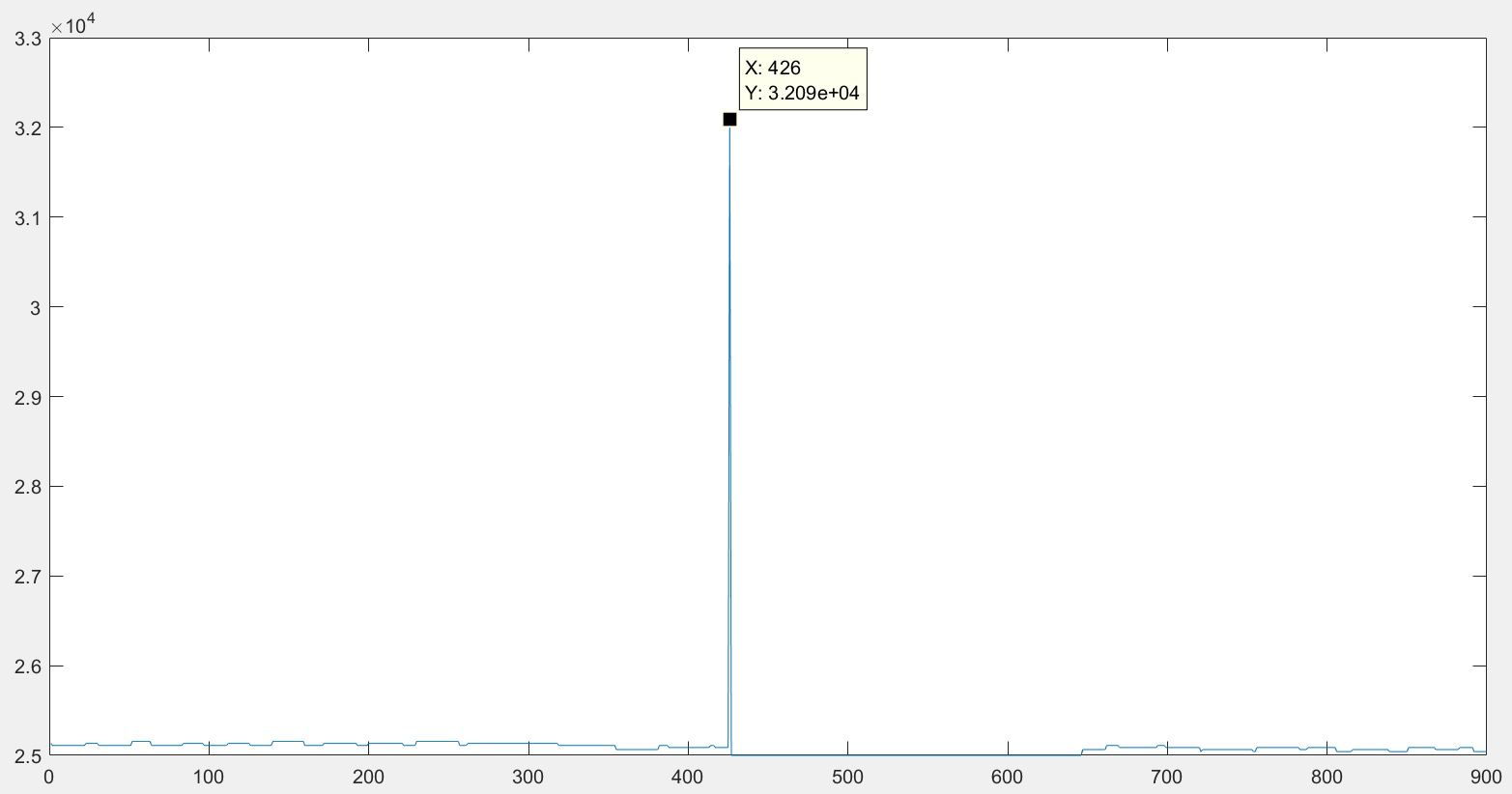
Pre\_bit\_cnt > 231

Count was saved on the pst\_trig.



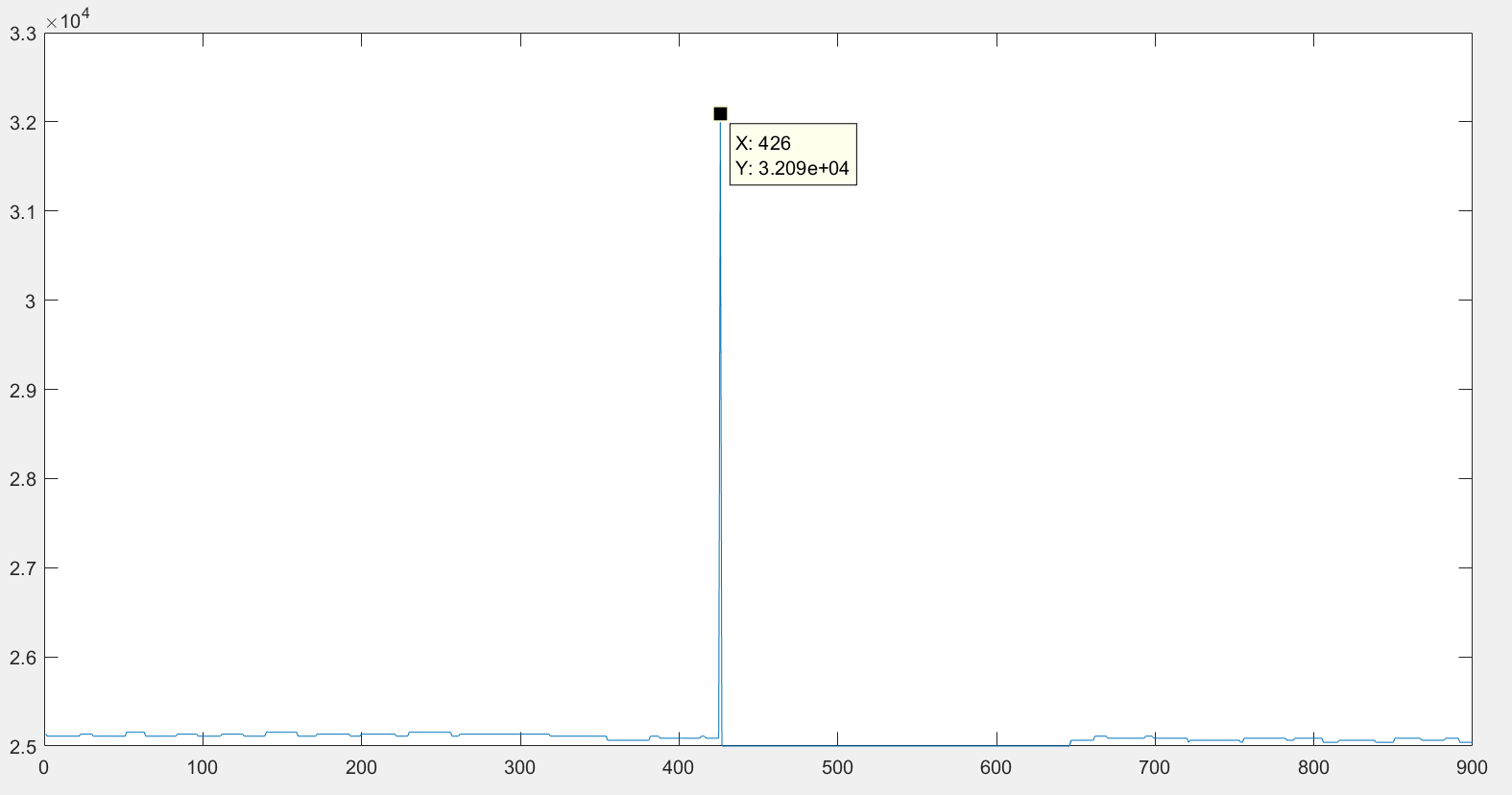
Pre\_bit\_cnt > 232

Count was saved on the pst\_trig.



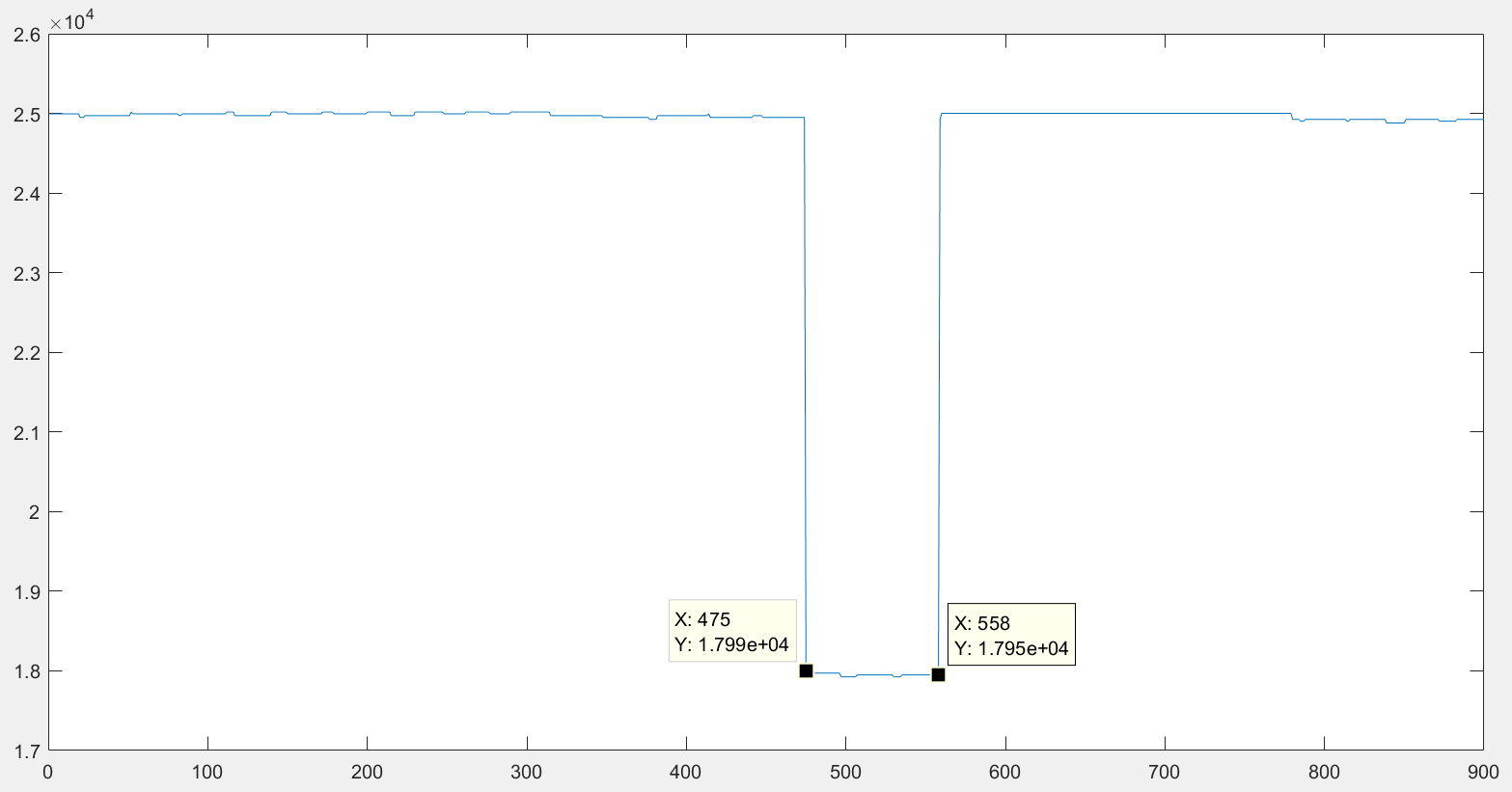
Pre\_bit\_cnt > 233

Count was saved on the pst\_trig.



Pre\_bit\_cnt > 234

Count was saved on the pst\_trig.



Pre\_bit\_cnt > 235

Count was saved on the pst\_trig.