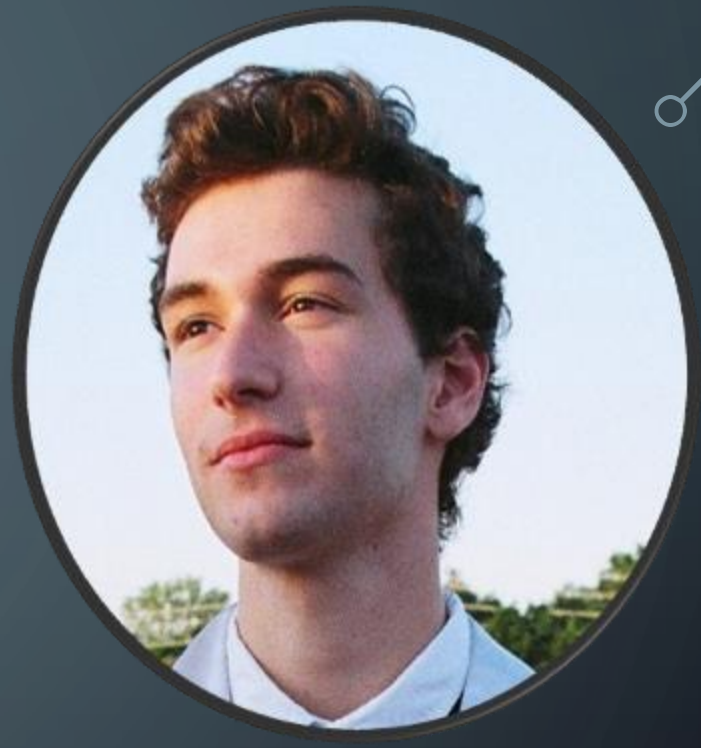
BRYAN GUNER ENGINEERING PORTFOLIO

TABLE OF CONTENTS

Contact Information:



Address: 150 Henley Place, Weehawken, NJ, 07086

LinkedIn: https://www.linkedin.com/in/bryan-guner-046199128/

Phone (mobile): 551-254-5505

Email(s):

bryan.guner@gmail.com

gunerb1@tcnj.edu

GitHub:

https://github.com/gunerb1

TECHNICAL SKILLS & SOFTWARE

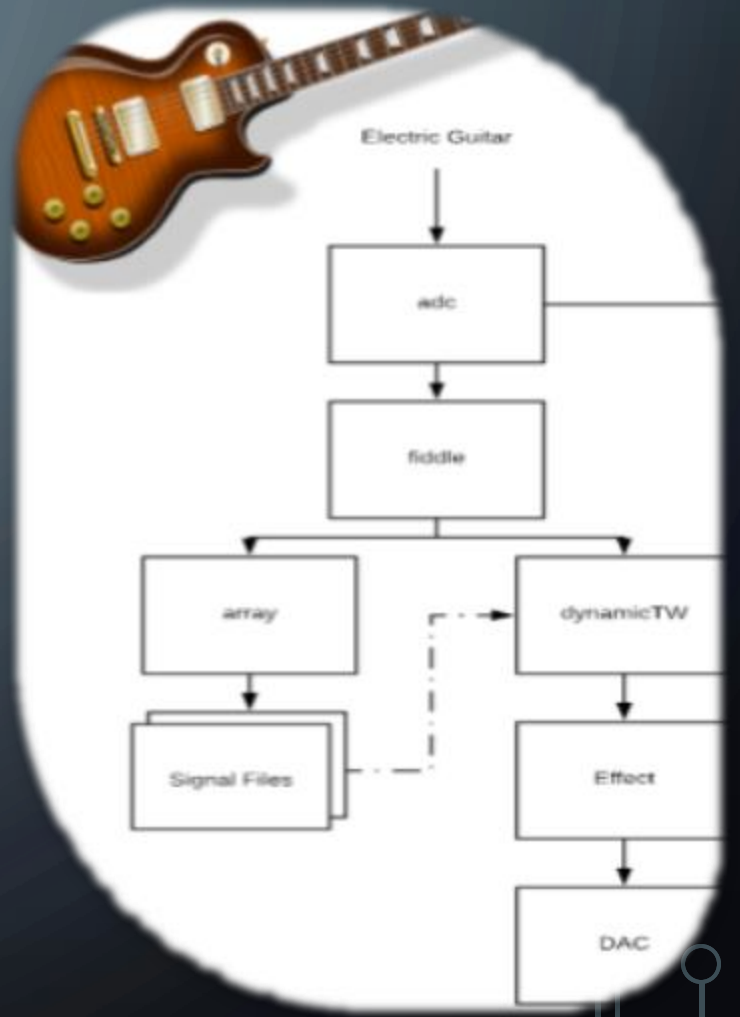
DYNAMIC TIME WARPING TRIGGERED GUITAR EFFECTS PLATFORM

Senior Design Project (TCNJ)

In live performance, guitar effect pedals are a versatile yet limiting asset, requiring presence of mind on the part of the performer. This platform offers an automatic solution to the restrictions that guitar effect pedals present.

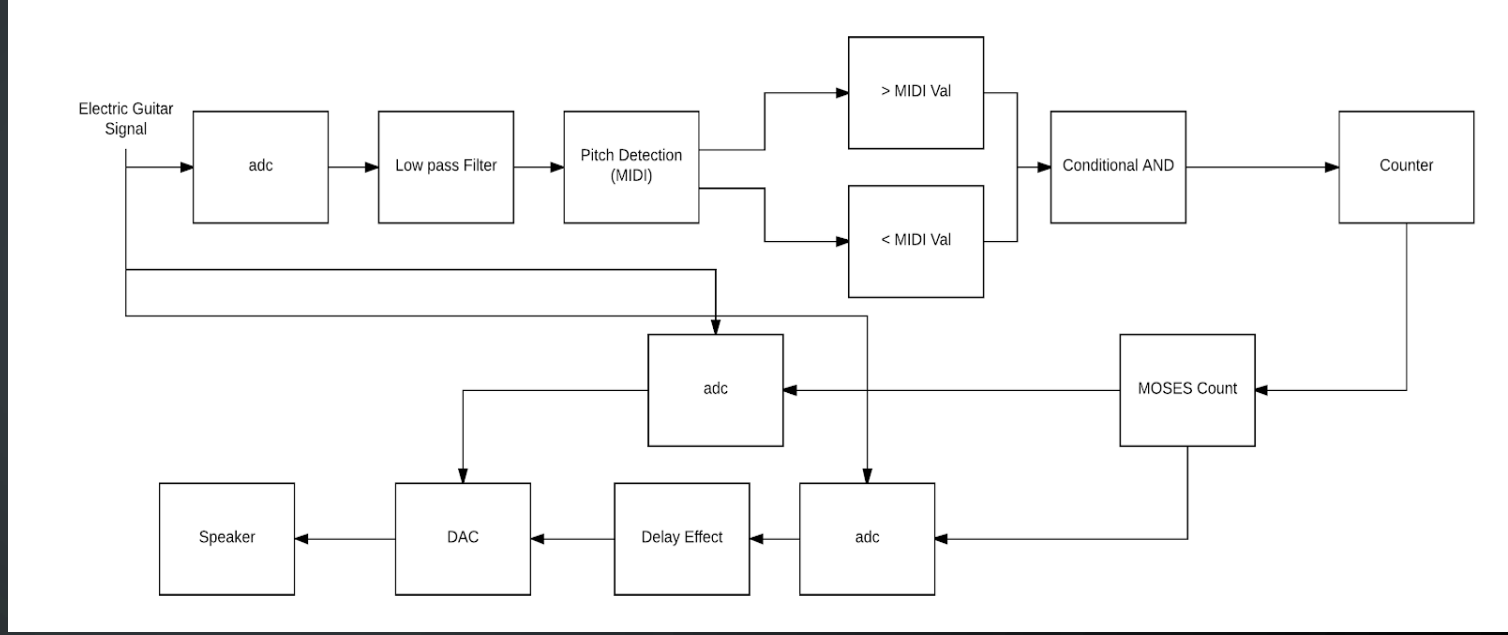
• System Architecture:

- read in a guitar signal during the 'learning' phase and isolate subsections of a performance needed to generate the DTW learned-threshold.
- then implement an analysis based on a modified dynamic time warping (DTW) algorithm, to compare the DTW cost function of incoming live audio against the cost-thresholds determined in the pre recording phase.



HOW IT WORKS:

This automation was achieved through the use of Pure Data, a GUI for audio manipulation applications, with embedded Python externals. When the algorithm detects a match, the platform runs the 'pure' digitalized audio signal through custom made PD effects patches!

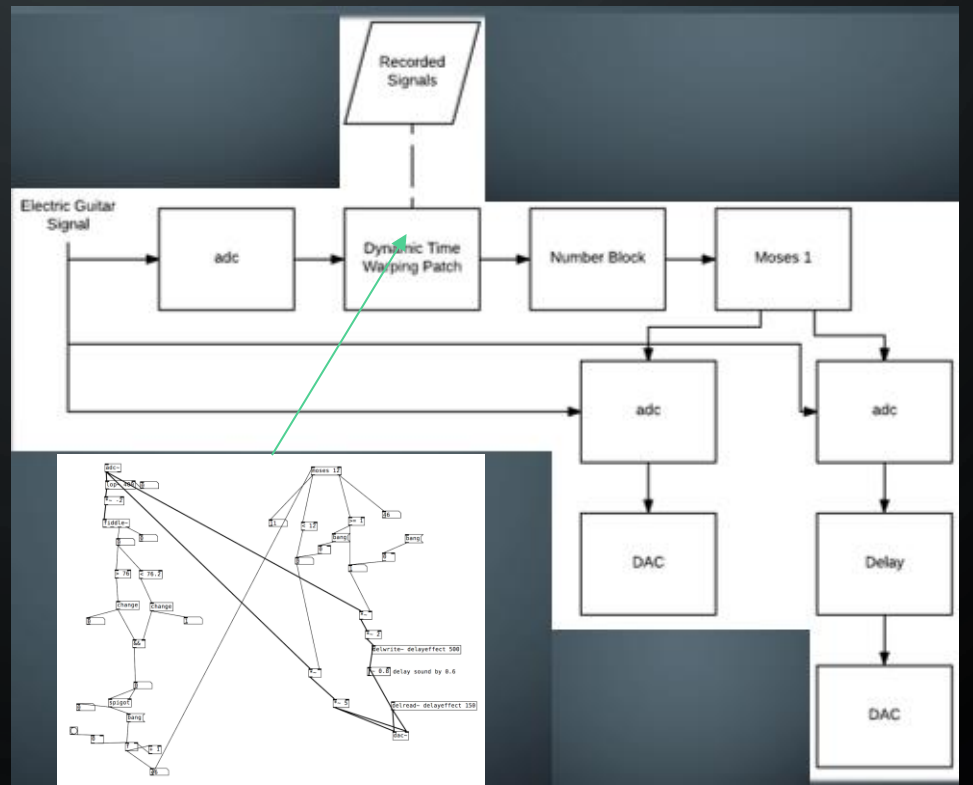


Dynamic Time Warping External:

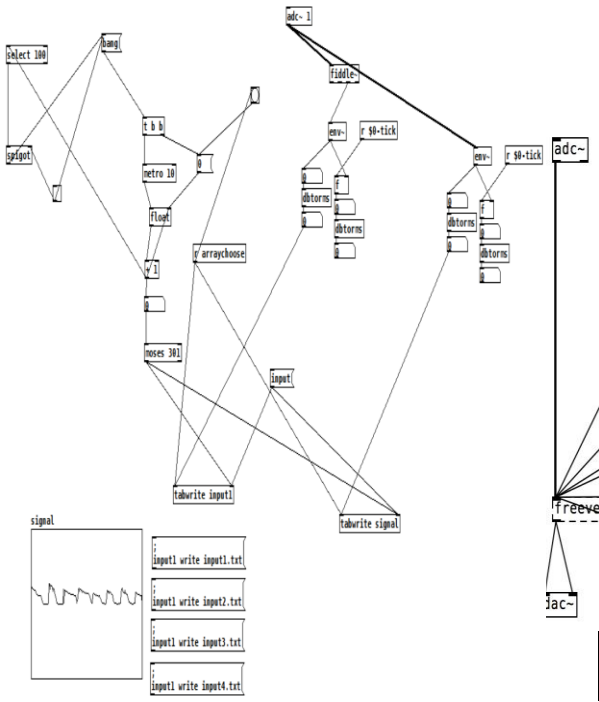
•Feed in two song performances for learning phase and obtain Least Cost Path (LCP) for each sub signal

•Compare Incoming live signal with sub signal of one of the recorded performance

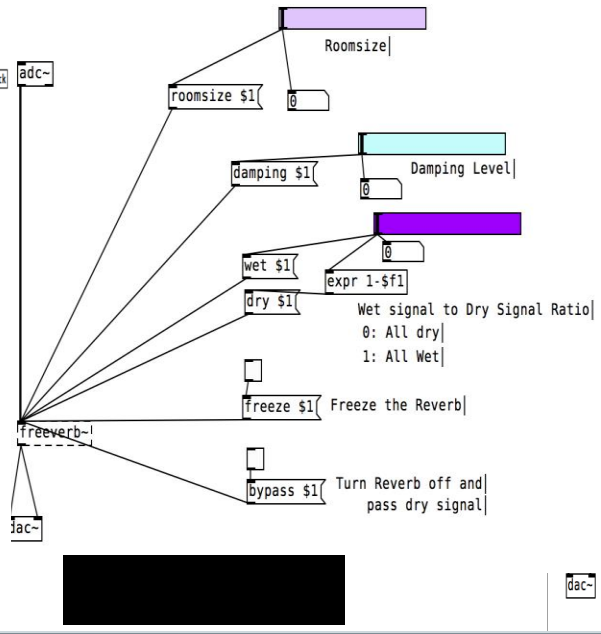
•When LCP value is less than or equal to the LCP obtained from learning phase, trigger guitar effect



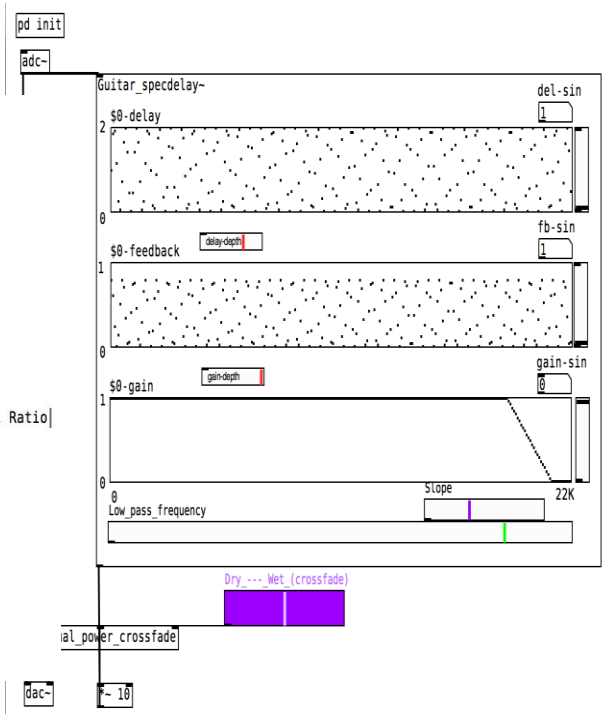
Recording (.wav) Patch:



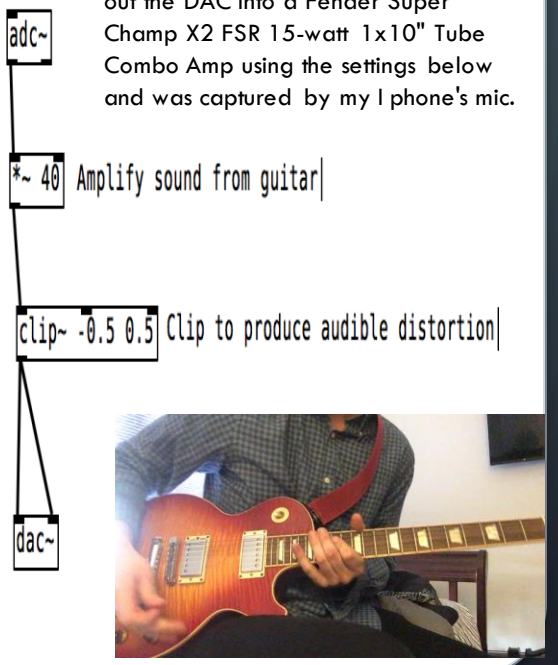
Reverb Patch



Spectral Delay Patch



(visible part of)Fuzz Patch

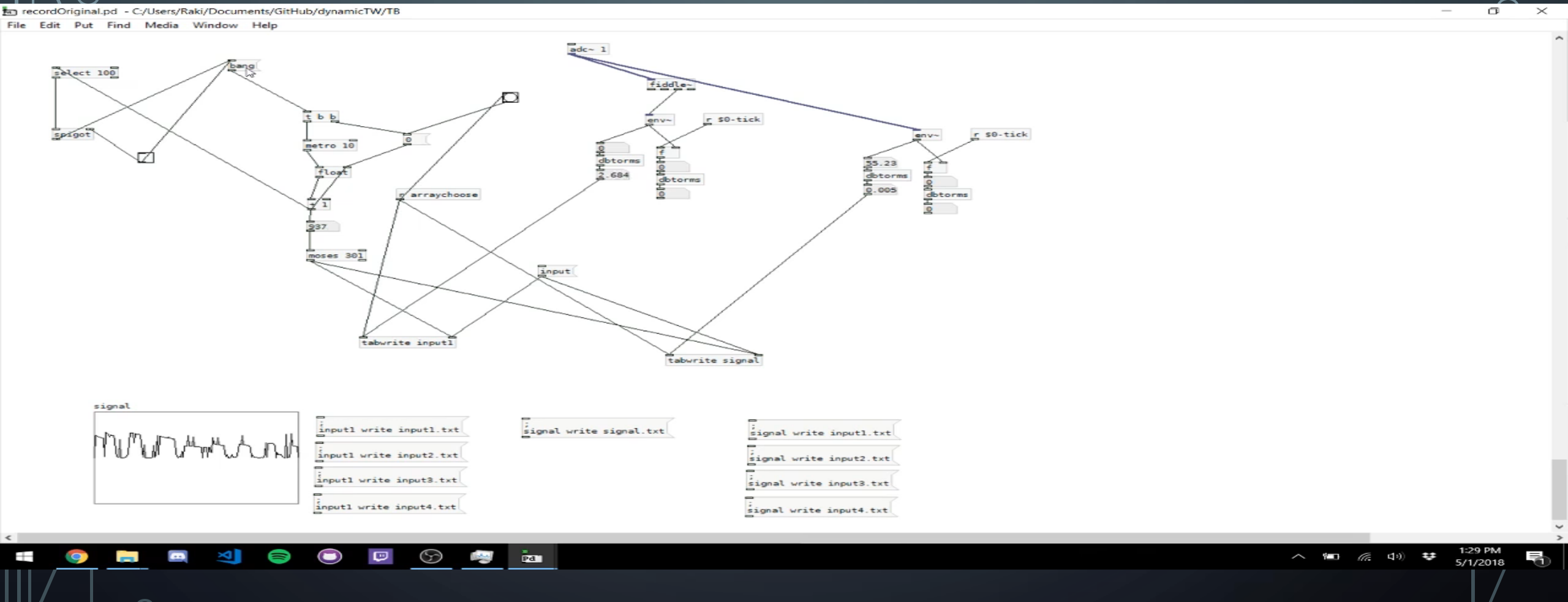


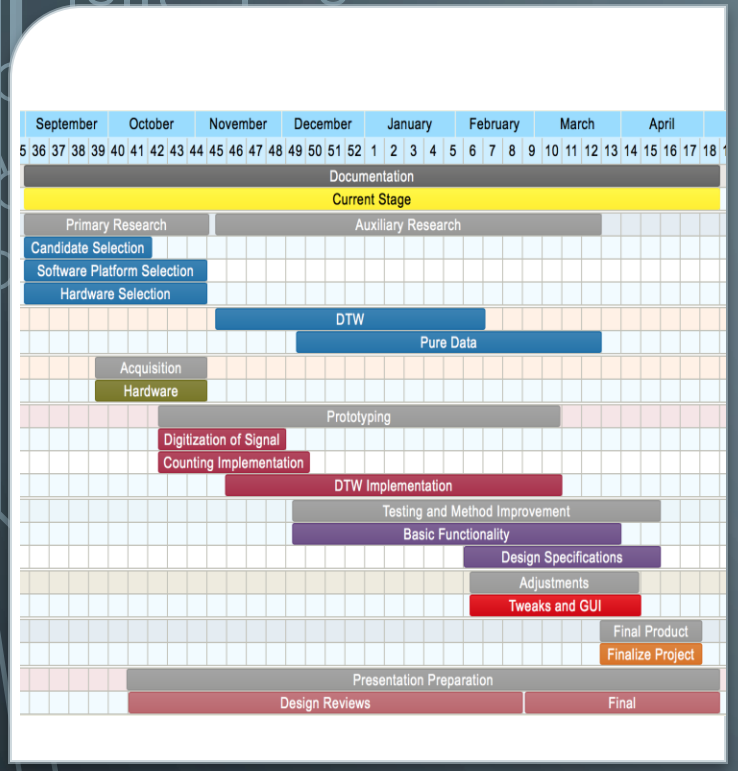
Signal sent DI into Scarlet 2i2 to be ported into pure data and then back out the DAC into a Fender Super Champ X2 FSR 15-watt 1x10" Tube Combo Amp using the settings below and was captured by my I phone's mic.

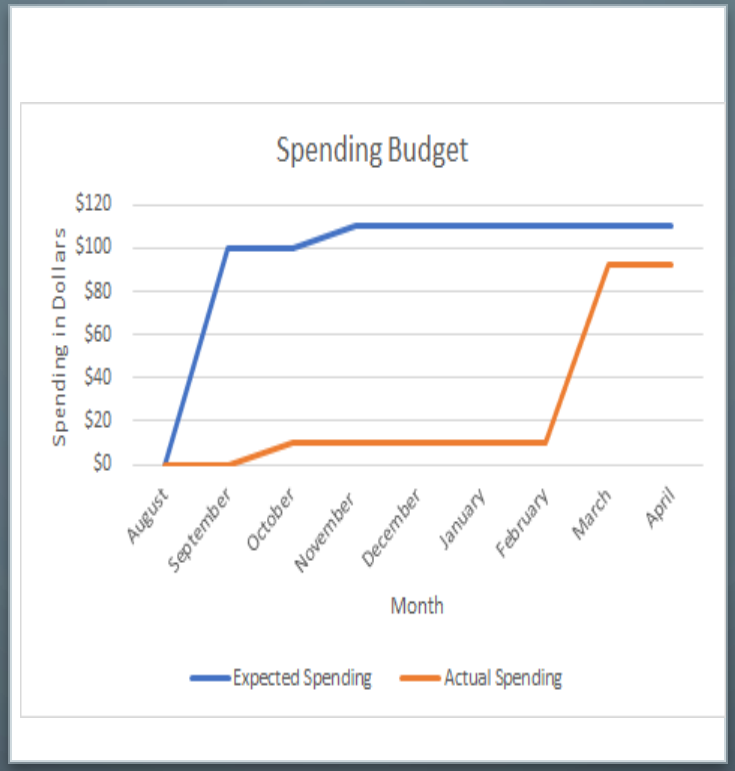
PURE DATA PATCH DEMOS:

PLATFORM DEMO (WAIT UNTIL 0:18 FOR AUDIO)

This demo uses the reverb patch (please forgive the quality of my guitar performance in this; at the time of this recording I hadn't slept in almost 48 hours in a desperate attempt to meet multiple concurrent project deadlines!!!)







DOCUMENTATION

Senior Design Thesis
Grade Awarded: A-

CODE @ EITHER:
HTTPS://GITHUB.COM/GUNERB1/TRIGGERED-GUITAR-EFFECTS-PLATFORM

OR (FOR MORE COMPREHENSIVE DOCUMENTATION REFER TO MY PARTNER RALPH QUNITO'S REPOSITORIES!)

HTTPS://GITHUB.COM/RAKIROAD/DYNAMICTW &

HTTPS://GITHUB.COM/RAKIROAD/TRIGGERED-GUITAR-EFFECTS-PLATFORM

PSOC PROGRAMMABLE MICROCONTROLLER GUITAR FIR DELAY PEDAL

BRYAN GUNER

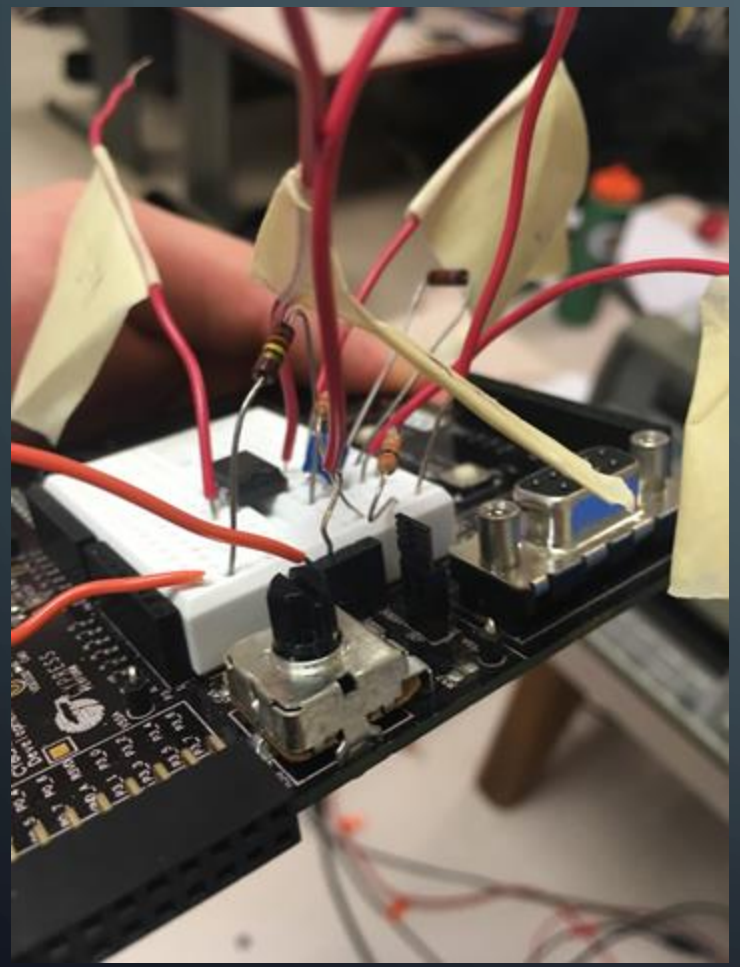
DESIGN ELEMENTS

- Analog to Digital Converter (taking analog guitar input, converting to digital to read and utilize)
- Rectifier & Pre-Amplifier (to accommodate the input constraints of the micro controller)
- Digital to Analog Converter (taking digital signal, converting to analog for output)
- Finite Impulse Response Digital Filter
- 2 Push Buttons

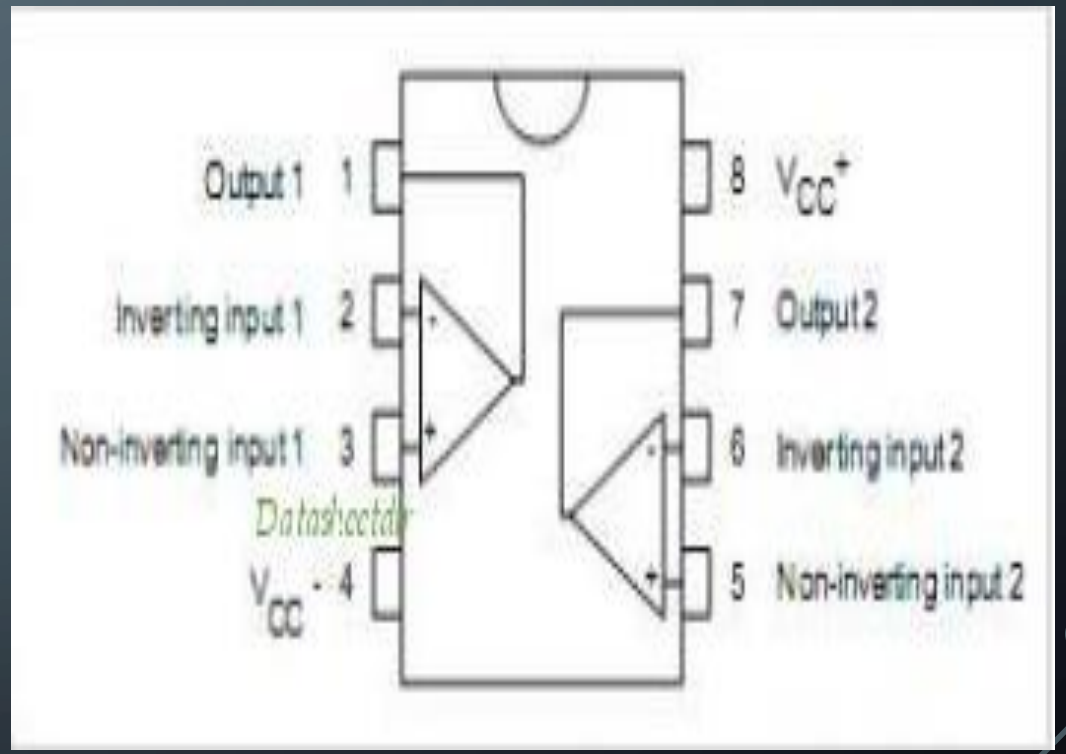
OBJECTIVE

- Design a guitar pedal to take a guitar signal as an input, and output the delayed and echoed sound numerous times
- The pedal must implement a preamplifier in order to manage the negative voltage input
- The pedal must poll for updated input data in order to produce a continuous output
- The pedal must use an ADC in order to read and utilize the input

CURRENT MODEL

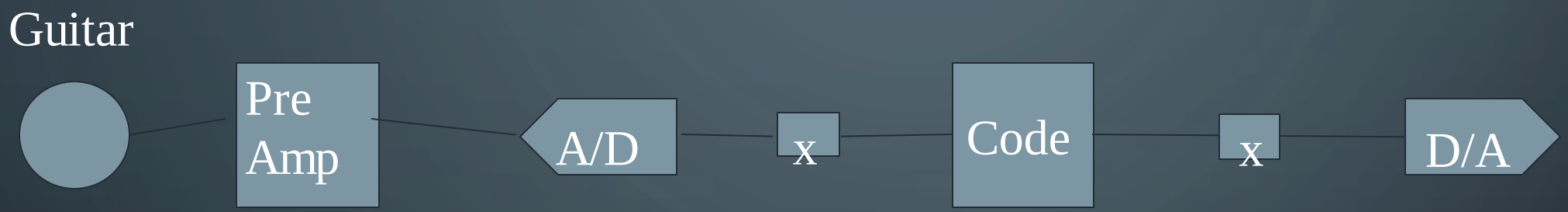


Physical Implementation of Circuit Design



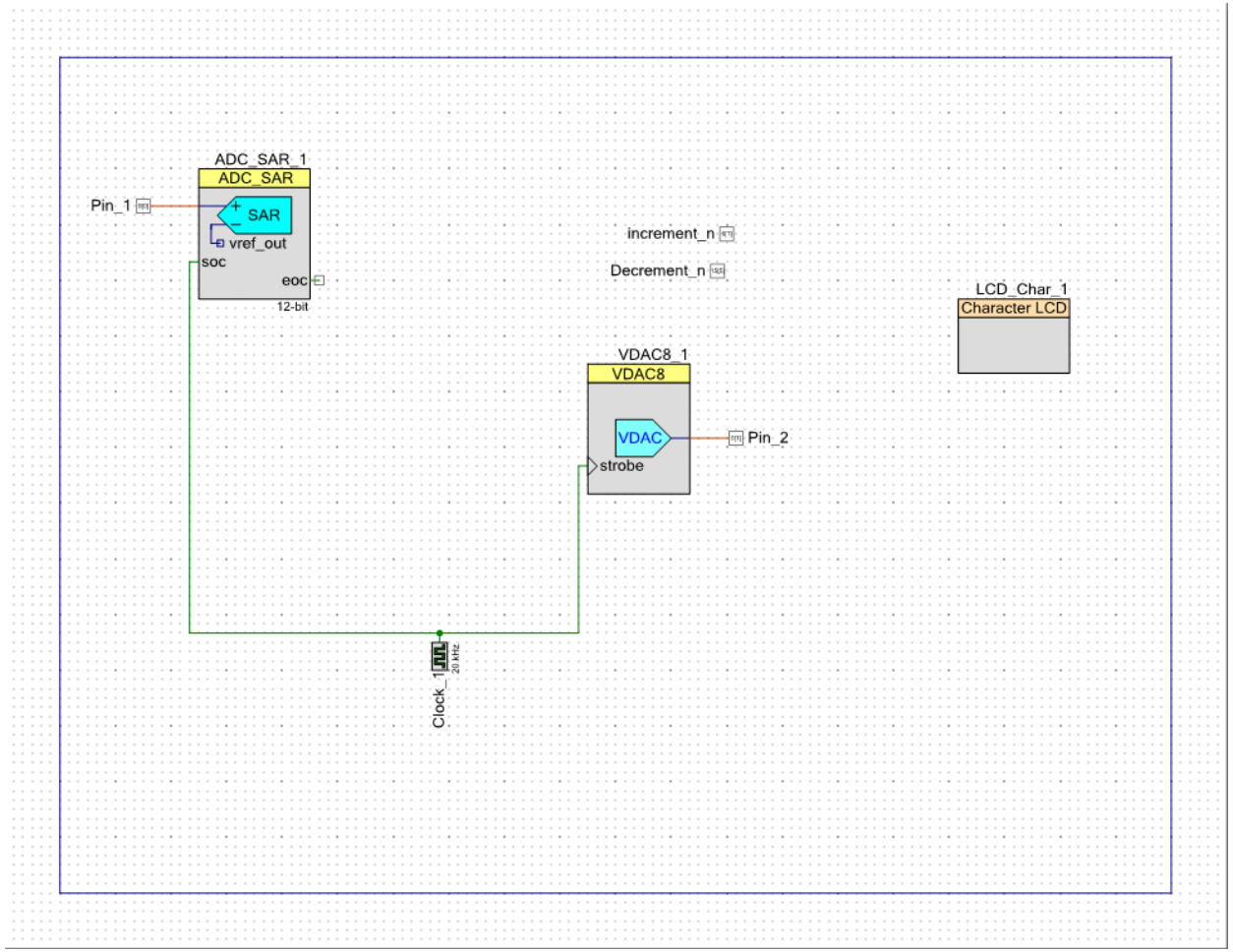
Dual Op-Amp (MC1458)

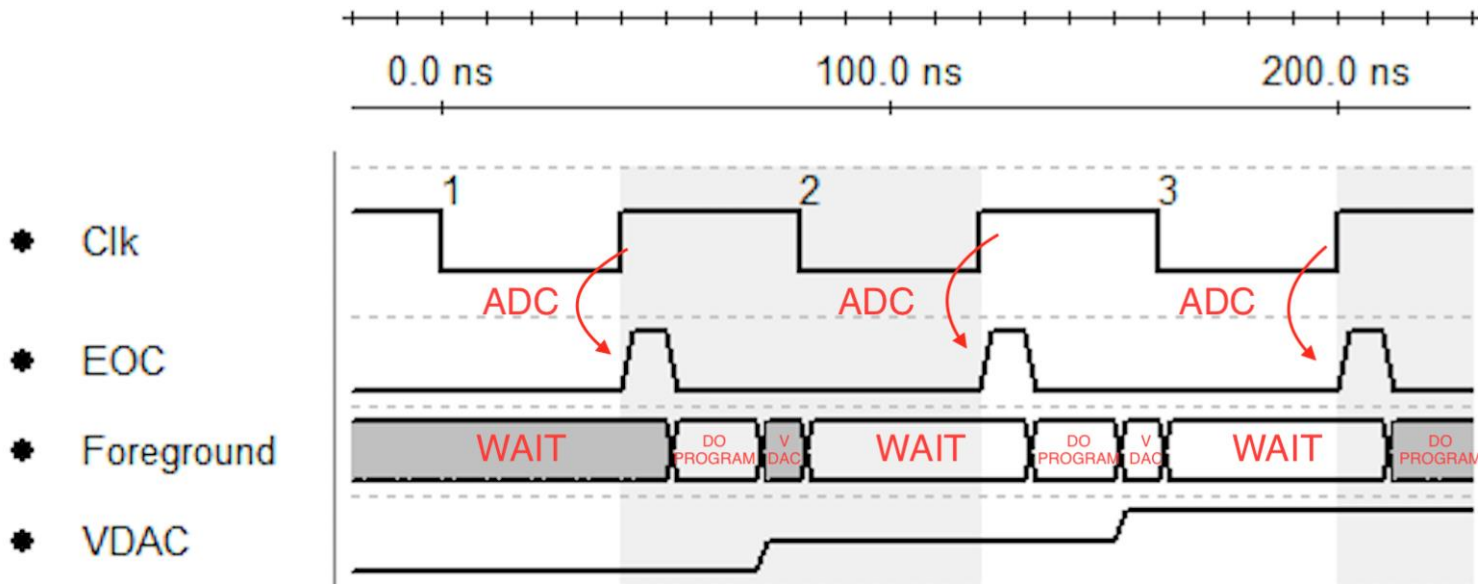
BLOCK DIAGRAM



Amplifier

PSOC SCHEMATIC





TIMING DIAGRAM

CURRENT MODEL CODE

```
int main()
{
    int maxs=20000;
    int sum=0;
    int windex=0;
    volatile uint16_t n=10;
    int delay=2000;
    int k;
    uint16_t x;
    uint16_t gsa [20000]={0};            //sample array initialized to 0


     CyGlobalIntEnable;             /* Enable global interrupts. */

     Clock_1_Start() ;
     ADC_SAR_1_Start();
     VDAC8_1_Start();
     LCD_Char_1_Start();
     LCD_Char_1_Position(0u, 0u);

     for(;;)
     {
     LCD_Char_1_ClearDisplay();
     LCD_Char_1_PrintNumber(n);                              //outputs n to display
     CyDelay(500);

         if( Decrement_n_Read() == 0 )                       // if pressed
         {
             CyDelay(500);                                   //to make sure increment/decrement event registers only once per press
             n=n-1;                                          //decrement
         }
         else if( increment_n_Read() == 0)
         {
            CyDelay(500);
             n=n+1;                                          //Increment
         }


         ADC_SAR_1_IsEndConversion(ADC_SAR_1_WAIT_FOR_RESULT);//polling
         x=ADC_SAR_1_GetResult16();
         gsa[windex]=x;                                      //indexing sample array
         sum=0;
         for(k=0;k<n;++k)
         {
             sum+=gsa[(windex+maxs-k*delay)%maxs];           //circular buffer

         }
         sum=sum>>4;
         sum=sum/n;                                         //Prevents increasing amplitude with every echo
         VDAC8_1_SetValue(sum);
         windex=(windex+1)%(maxs);
         }
}

```

MATLAB SIMULATION

- MATLAB was used to debug and simulate the guitar delay pedal
- Utilized FIFO (first in first out), delay and echo

FIFO.m | Delay_Model.m | Echo.m

```
FIFO ('init');
fs = 20000;
f0=440;
 delay = 6000;
 n_echos = 6;
x=(1:10*fs);
x=exp(-1.*x./2000).*cos(2*pi()*fs/f0*(1+x/100000).*x);
plot(x(1:1000))
%x(1000:end) = 0;
%sound(x)

for n=1:20000 %assuming 10 seconds * 20,000
    y(n)=Echo(x(n),delay,n_echos);
end

    sound(y);
    plot(y);
```

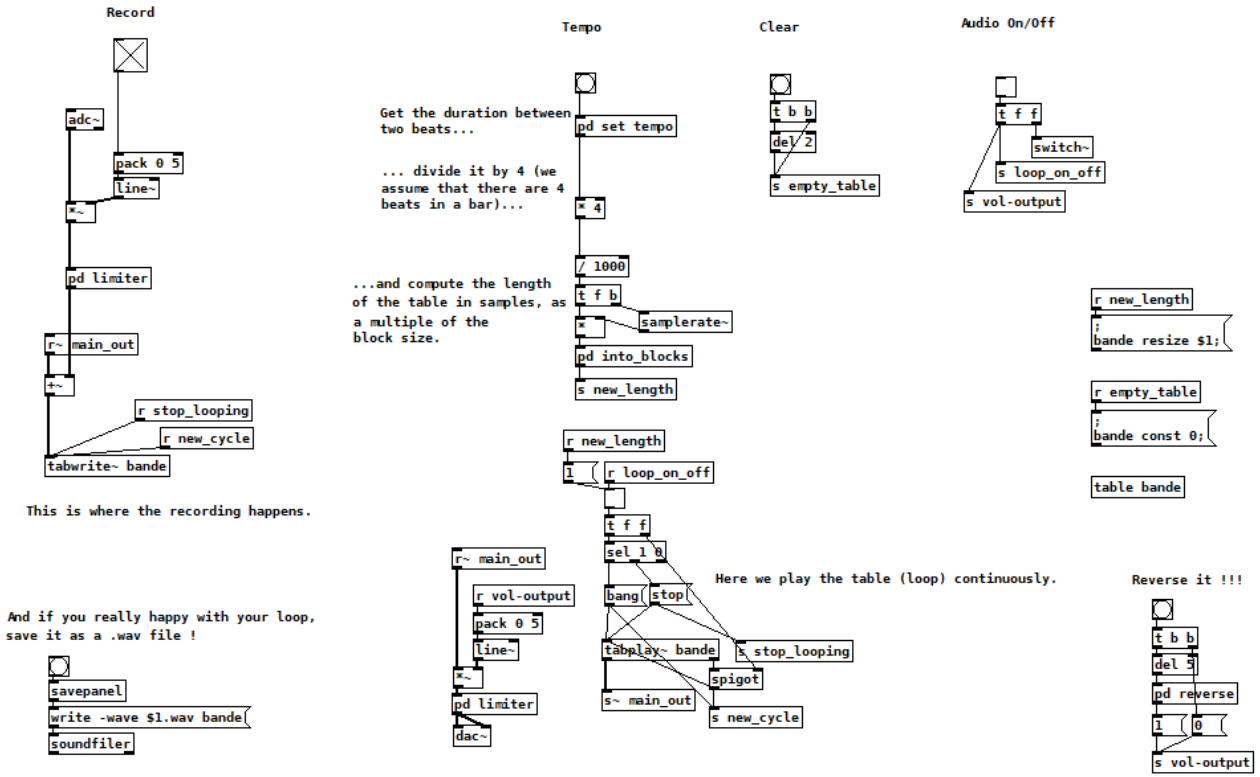
FIFO.m | Delay_Model.m | Echo.m

```
function y = Echo(x, delay, n_echos )



  sum = 0;
 FIFO('write', x);
      for i = 1:n_echos
          sum = sum + FIFO('read', delay * i);
      end
   y = sum;
end
```

FIFO.m | Delay_Model.m | Echo.m

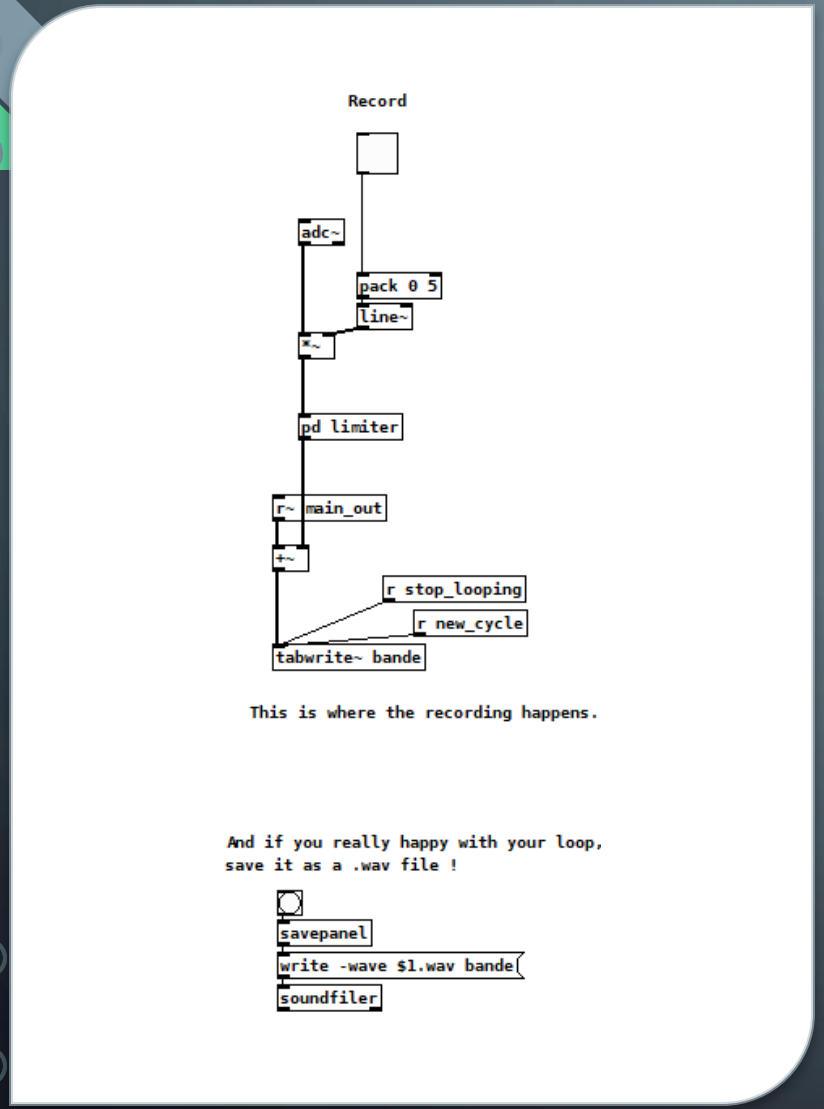
```
function result = FIFO (command, arg)
    if strcmp(command, 'read')
        result = read_fifo(arg);
    else
        if strcmp(command, 'write')
            write_fifo(arg);
        else
            if strcmp(command, 'init')
                init_fifo();
            end
        end
    end
end

function init_fifo()
global fifo_buf write_ptr MAX_DELAY; % globalizing variables
MAX_DELAY = 20000;%Max delay of 20kHz

fifo_buf = zeros(MAX_DELAY,1);%buffer = array of all zeros

write_ptr = 1;
end
function val = read_fifo(delay)% Val = fifo output
   global fifo_buf write_ptr MAX_DELAY;

    % Read delayed values
    % Delay read_ptr by an amount of samples, based on 'delay'
    read_ptr = mod((write_ptr - 2) - delay + MAX_DELAY, MAX_DELAY)+1 ;

    val = fifo_buf(read_ptr);

end

function write_fifo( val )%using output of read_fifo
global fifo_buf write_ptr MAX_DELAY;

    fifo_buf(write_ptr) = val;
    write_ptr = write_ptr + 1;%increment write_ptr by 1

    if (write_ptr > MAX_DELAY)%Once write_ptr is greater than max delay, restart
        write_ptr = 1;
    end
end
```

PURE DATA LOOPER



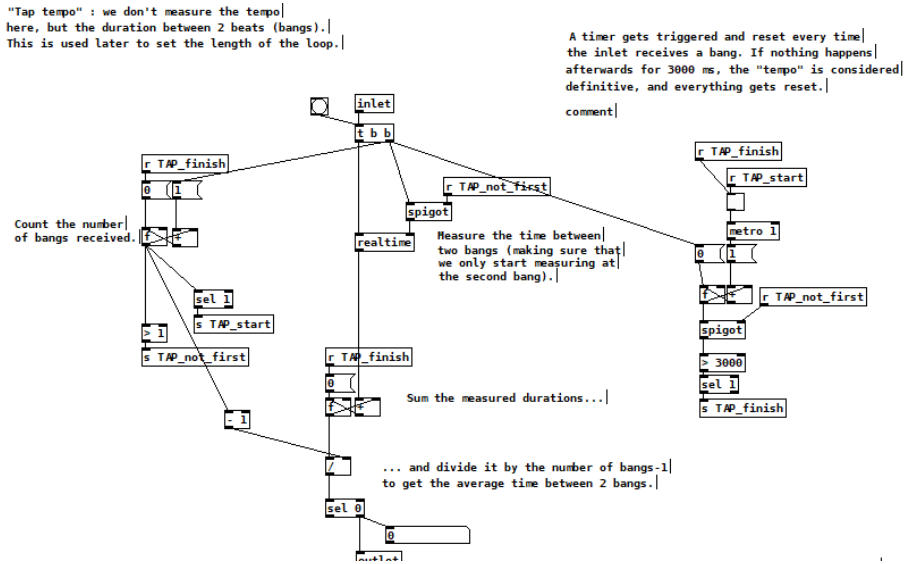
Features:

- tap tempo
- Limiter on both the input and the output to prevent intermodulation distortion produced by superimposed layers.
- “Reverse” function for fun!



BASIC OPERATION

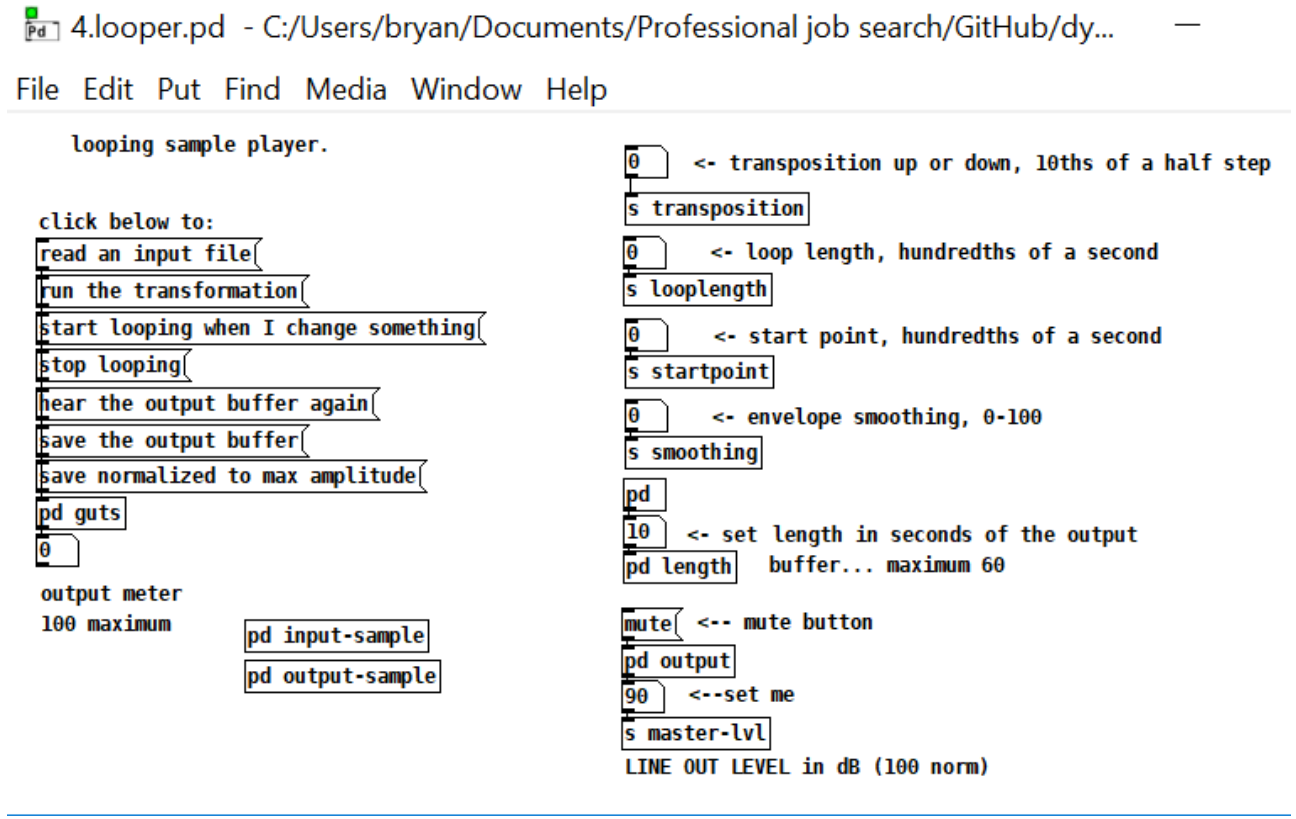
- The loop is stored in a table in Pd, which is played repeatedly.
- While this table is being played, the output and the current input is also written to it.
- If there is no input the loop copied on itself.



DESIGN CONSIDERATIONS

- Pure Data processes the audio in blocks of samples (64 samples by default).
- Any audio event occurring during a block will only be taken into account at the beginning of the next block.
- The table's length must thus be a multiple of the block size (64 in my patch), so that the end of the last block of the loop corresponds exactly to the beginning of the first block.
- If the table's length is not a multiple of the block size, audible pops and clicks will appear while looping, and gradually turn into a very annoying noise.

PURE DATA SAMPLER



Modeling and Discrete-Logic Control of a Quadruple-Tank System

Bryan Guner[1] and Sean Fernandez[1]; Advisor: Dr. Ambrose Adegbege[1]

[1]Department of Electrical and Computer Engineering, The College of New Jersey, Ewing, NJ

Abstract

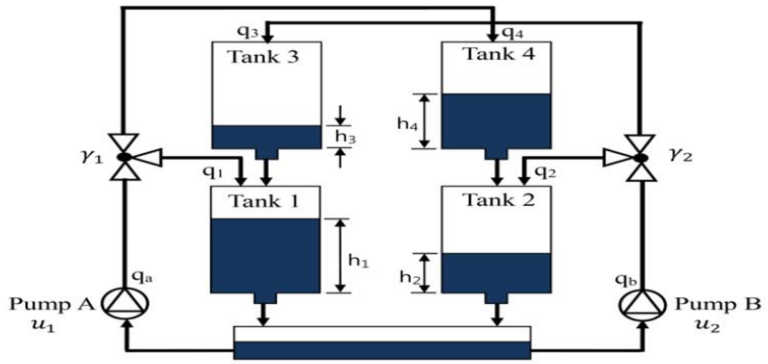
The goal of this design assignment was to develop and simulate a mathematical model of a quadruple-tanks system and subsequently implement discrete-logic control using the LabVIEW software package. The purpose was to exercise our knowledge of the differential equations that represent such a system, and to convert them into linearized state-space and transfer function descriptions that could be more easily used to simulate said system in LabVIEW.

Introduction

The aforementioned quadruple tank system consists of four water tanks and two pumps that form a two-input two-output system. The intention is to control the level of the water in the bottom two tanks while the pumps delivers water to the top two tanks that trickle down into the bottom two via two outlets situated at the bottom of the top two tanks. The inputs are the control voltages sent to the pumps and the outputs are the sensor voltages corresponding to the water levels in the lower tanks. A mathematical model is developed using a combination of the mass balance equation, Bernoulli's equation and the derived flow coefficient taking into account the proportionality splitting in the pump system.

The system is first modeled by a set of nonlinear differential equations which are then linearized about an operating point that is comprised of a water level height argument and the voltage to the pumps.

Schematic of 4 Tank System:



Nonlinear Equations:

$$\frac{dh_1}{dt} = -\frac{a_1}{A_1}\sqrt{2gh_1} + \frac{a_3}{A_1}\sqrt{2gh_3} + \frac{\gamma_1 k_1 v_1}{A_1}$$

$$\frac{dh_2}{dt} = -\frac{a_2}{A_2}\sqrt{2gh_2} + \frac{a_4}{A_2}\sqrt{2gh_4} + \frac{\gamma_2 k_2 v_2}{A_2}$$

$$\frac{dh_3}{dt} = -\frac{a_3}{A_3}\sqrt{2gh_3} + \frac{(1-\gamma_2)k_2 v_2}{A_3}$$

$$\frac{dh_4}{dt} = -\frac{a_4}{A_4}\sqrt{2gh_4} + \frac{(1-\gamma_1)k_1 v_1}{A_4}$$

$$\hat{y}_1 = k_c h_1, \qquad \hat{y}_2 = k_c h_2$$

Linear Equations:

$$\frac{dx_1}{dt} = -\left(\frac{a_1}{A_1}\sqrt{\frac{g}{2h_1^0}}\right)x_1 + \left(\frac{a_3}{A_1}\sqrt{\frac{g}{2h_3^0}}\right)x_3 + \left(\frac{\gamma_1 k_1}{A_1}\right)u_1$$

$$\frac{dx_2}{dt} = -\left(\frac{a_2}{A_2}\sqrt{\frac{g}{2h_2^0}}\right)x_2 + \left(\frac{a_4}{A_2}\sqrt{\frac{g}{2h_4^0}}\right)x_4 + \left(\frac{\gamma_2 k_2}{A_2}\right)u_2$$

$$\frac{dx_3}{dt} = -\left(\frac{a_3}{A_3}\sqrt{\frac{g}{2h_3^0}}\right)x_3 + \left(\frac{(1-\gamma_2)k_2}{A_3}\right)u_2$$

$$\frac{dx_4}{dt} = -\left(\frac{a_4}{A_4}\sqrt{\frac{g}{2h_4^0}}\right)x_4 + \left(\frac{(1-\gamma_1)k_1}{A_4}\right)u_1$$

$$y_1 = k_c x_1, \qquad y_2 = k_c x_2$$

Design/Methods

State-Space Configuration:

$$\frac{dx}{dt} = \begin{bmatrix} -0.015948 & 0 & 0.041859 & 0 \\ 0 & -0.01107 & 0 & 0.033341 \\ 0 & 0 & -0.041859 & 0 \\ 0 & 0 & 0 & -0.033341 \end{bmatrix} x(t) + \begin{bmatrix} 0.08325 & 0 \\ 0 & 0.062813 \\ 0 & 0.047857 \\ 0.031219 & 0 \end{bmatrix} u(t)$$

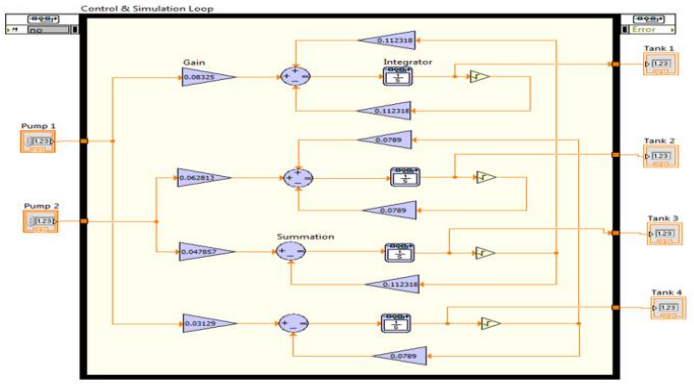
$$y(t) = \begin{bmatrix} 0.5 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 \end{bmatrix} x(t) + \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} u(t)$$

Transfer Function Configuration:

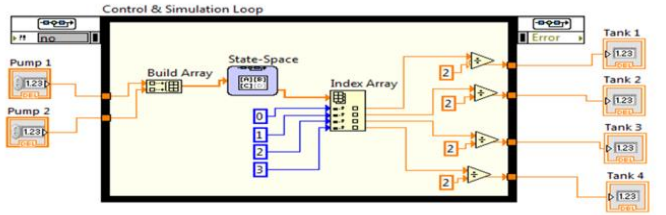
$$H(s) = \begin{bmatrix} \frac{2.6}{62s+1} & \frac{1.4}{2700s^2+120s+1} \\ \frac{1.41}{1426s^2+85s+1} & \frac{2.8}{90s+1} \\ 0 & \frac{0.572}{23.89s+1} \\ \frac{0.4682}{23.993s} & 0 \end{bmatrix}$$

LabVIEW Implementation:

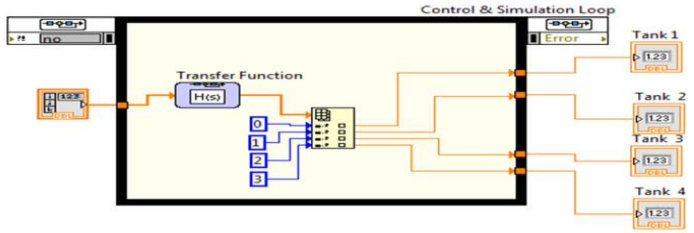
Nonlinear Model sub-VI:



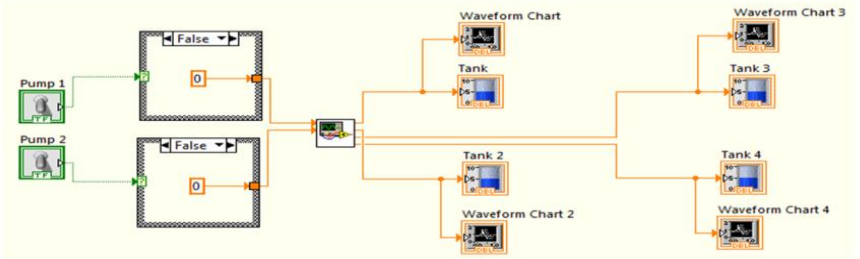
Linearized State-Space Model sub-VI:



Linearized Transfer Function Model sub-VI:

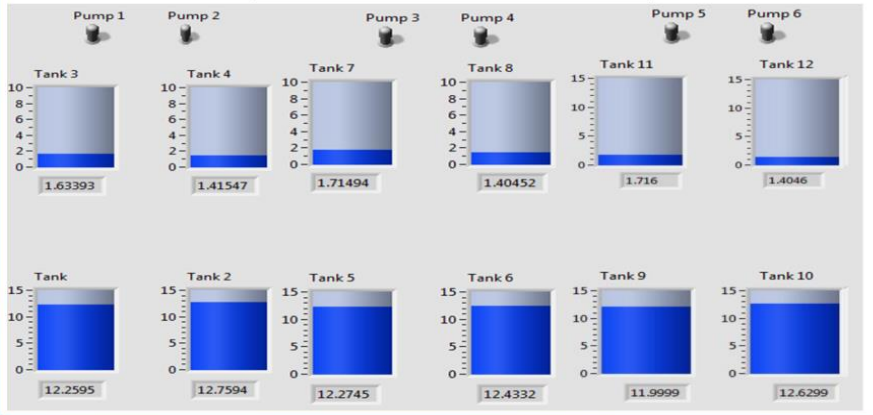


Quadruple Tank-System Block Diagram:



Results

Output Water Levels Front Panel:



Conclusion

- Purpose was to implement and simulate both state space and transfer function to model behavior of system
- Valuable learning experience in linearization of discrete control of systems
- Overall, students successfully modeled a quadruple-tank system utilizing LabVIEW
- Learned about the use of subVI's to reduce complexity and size of code
- Close approximation to working experience of a control systems engineer

Design of a Balance and Swing-up Control System for a Rotary Inverted Pendulum

Bryan Clum Guner*, Christopher Hingston Tenev*
*Department of Electrical Engineering
The College of New Jersey, Ewing, NJ

Abstract

The purpose of this experiment was to design and implement a state-feedback control system to control a rotary inverted pendulum system. This exercise served to introduce students to the process of linearizing nonlinear equations of motion in order to obtain the linear state-space representation of the pendulum system, and to demonstrate the effectiveness of single-loop feedback control in electromechanical application. A state-space model of the Quanser SRV02 Rotary Pendulum System was developed, and controls for directing the system's balance and vertical orientation were designed.

Introduction

In these experiments, a hardware system was modelled in MATLAB and controlled in Simulink using a single-loop feedback system.

The hardware system that was modelled and controlled consists of an actuated rotary arm, connected to a free-swinging, invertible pendulum arm; the positions of both arms are determined in real time by encoders. To provide a reliable and consistent mathematical model of the hardware system, time-invariant state-space representation was employed, which formed the basis of all the implemented control.

The system is required to balance the pendulum arm vertically, and to swing the pendulum up from the downward-hanging position. Control of the pendulum system was implemented in Simulink, by applying negative feedback to the motor driver; the feedback gain required to control the system was calculated in real time using the previously determined state-space model and the principles of pole placement, according to the desired control type.

Methods

Modelling

- State-space representation of the hardware components allows for description of the system with time-invariant differential equations.
- Time-invariant model allows for precise control, regardless of component position, speed, or acceleration.
- State-space model is used to continuously determine the control gain of the control voltage which drives the system's motor.

ROTPEN module

- State-space system model is implemented in a Simulink block to allow integration into larger systems within Simulink.
- Module reads encoder values from rotary and pendulum arms, and outputs rotary arm angle, ϑ, "raw" pendulum angle, α_r, and pendulum angle with respect to vertical, α.
- Negative inverse of control voltage drives motor to counteract falling of pendulum.

Balance control

- Signal generator provides vector state which instructs pendulum to remain within 20° of vertical.
- Multiport switch enables balance control when pendulum is within 20° of vertical; input voltage is zero when pendulum is further than 20° from vertical, and equals system error otherwise.
- ROTPEN module counteracts error by applying negative inverse of system error to the motor.

Swing-up control

- To allow for swing-up control, input voltage must be switched between error and energy-based control.
- Swing-up control must constantly move the pendulum with enough force to counteract gravity without overshooting the desired position.
- Friction must remain constant to avoid introduction of position- and velocity-dependent damping.

$$\begin{bmatrix}\dot{\vartheta}\\ \dot{\alpha}\\ \ddot{\vartheta}\\ \ddot{\alpha}\end{bmatrix} = \begin{bmatrix}0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\\ 0 & \frac{0.25L_p^2L_r g m_p^2}{J_rJ_p+0.25m_pJ_rL_p^2+J_pm_pL_r^2} & -\frac{B_r(J_p+0.25m_pL_p^2)}{J_rJ_p+0.25m_pJ_rL_p^2+J_pm_pL_r^2} & -\frac{0.5B_pL_pL_rm_p}{J_rJ_p+0.25m_pJ_rL_p^2+J_pm_pL_r^2}\\ 0 & \frac{0.5L_pm_pg(J_r+m_pL_r^2)}{J_rJ_p+0.25m_pJ_rL_p^2+J_pm_pL_r^2} & -\frac{0.5B_pL_pL_rm_p}{J_rJ_p+0.25m_pJ_rL_p^2+J_pm_pL_r^2} & -\frac{-(J_r+m_pL_r^2)B_p}{J_rJ_p+0.25m_pJ_rL_p^2+J_pm_pL_r^2}\end{bmatrix}\begin{bmatrix}\vartheta\\ \alpha\\ \dot{\vartheta}\\ \dot{\alpha}\end{bmatrix} + \begin{bmatrix}0\\ 0\\ \frac{0.25m_pL_p^2+J_p}{J_rJ_p+0.25m_pJ_rL_p^2+J_pm_pL_r^2}\\ \frac{L_pL_rm_p}{J_rJ_p+0.25m_pJ_rL_p^2+J_pm_pL_r^2}\end{bmatrix}u(\tau)$$

Figure 1: State-space model of inverted rotary pendulum system. Theta values refer to the pendulum, and alpha values refer to the rotary arm.

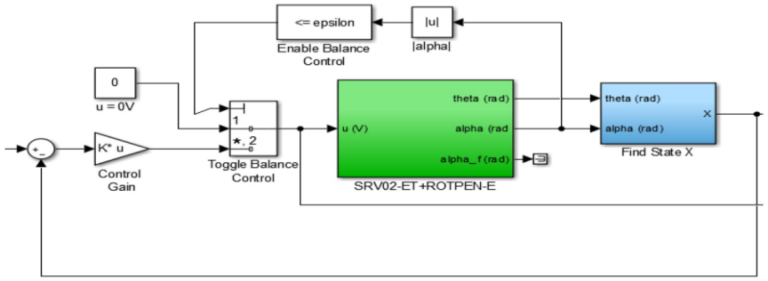


Figure 2: Balance control block diagram. The input to the rotary pendulum module (shown in green) is the error of the system; the rotary pendulum module takes the negative inverse of this input signal and applies it to the rotary arm motor to counteract the error. The switch enables negative feedback and control gain when the pendulum is within 20° of vertical.

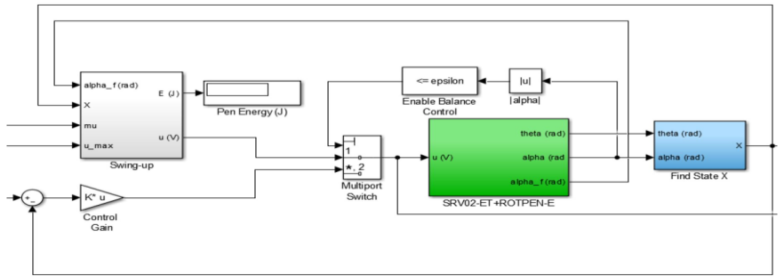


Figure 3: Block diagram showing implementation of rotary pendulum model with both balance and swing-up control. By replacing the 0 V input block in Figure 2 with the constant-energy swing-up control module shown here, the pendulum arm is swung upwards when it is further than 20° from vertical. When the pendulum arm is within 20° of vertical, the multiport switch enables the balance control which keeps the pendulum vertical.

Results

- State-space model of the rotary inverted pendulum system was devised, and the result was confirmed algebraically (see *Figure 1*).
- Simulink models were successfully implemented, and appropriate communication with hardware system was achieved.
- Some level of balance control was achieved, but the control gain appeared too high; motor arm over-compensated for the tilting of the pendulum arm.
- Swing-up control was attempted, but again, gain was too high.
- Manual gain adjustment was attempted, but the complexity of adjusting a vector quantity prevented full balance control from being achieved.

Conclusions

Despite setbacks and a lack of fully optimized balance and swing-up controls, the design successfully demonstrated the modelling of a hardware system in Simulink and MATLAB using state-space representation of the system and its output, as well as the control of such a model within the MATLAB/Simulink environment. State-space representation provided a model that could reliably describe the system regardless of the physical state of the hardware, and balance and swing-up controls were achieved through the implementation of simple feedback systems, the gain of which was determined in real time by the state-space system model and the physical status of the system.

It is believed that the error in implementing the controls was due to incorrect implementation of the state-space system model in MATLAB. The derivation of the state-space matrices was checked and re-checked, and the matrices are believed to be accurate. The error is thought to lie in the computation of the system's closed-loop poles, which are used by MATLAB to continuously calculate the system's control gain. Unfortunately, troubleshooting and optimization were not executed quickly enough to achieve the desired results within the time allotted for the experiments. Given more time to troubleshoot the MATLAB operations, it is likely that optimal performance would be achieved.

State-space modelling of hardware systems, in conjunction with single-loop feedback control systems implemented in Simulink, provide simple, yet powerful, tools for observing and controlling hardware systems mathematically. As any system may be represented by a single-loop feedback system, these methods may be used to model and control virtually any control system. The use of alternative control gain calculation methods, along with alternative system modelling techniques, such as with solid modelling CAD software, further expand the functionality and versatility of these design techniques.

References

[1] Nise, Norman S. *Control Systems Engineering.* 7th ed. New York: John Wiley, 2000. Print.

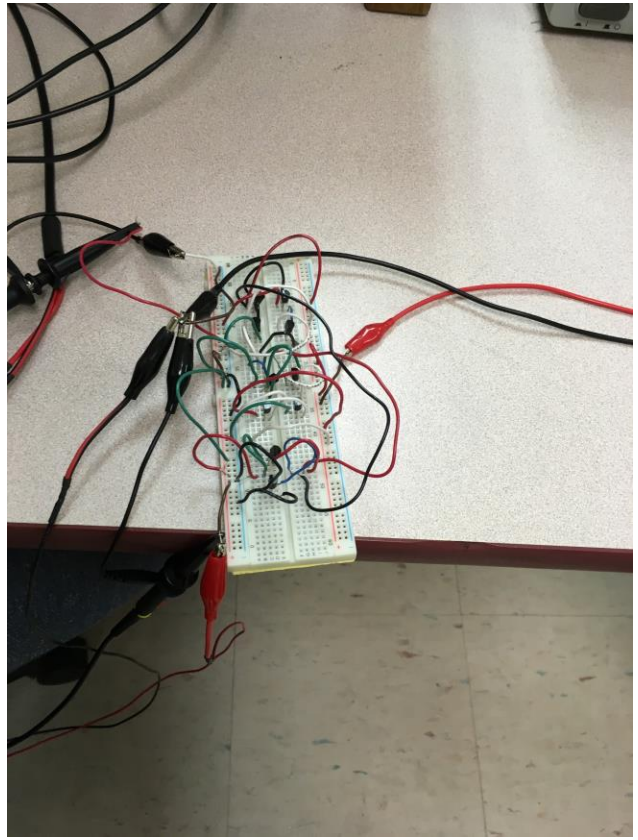
XOR GATE USING CMOS LOGIC

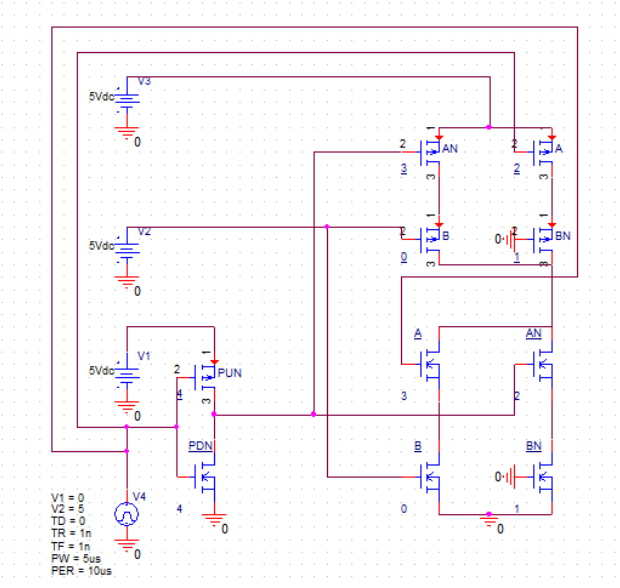
XOR GATE BACKGROUND

A	B	Out
0	0	0
0	1	1
1	0	1
1	1	0

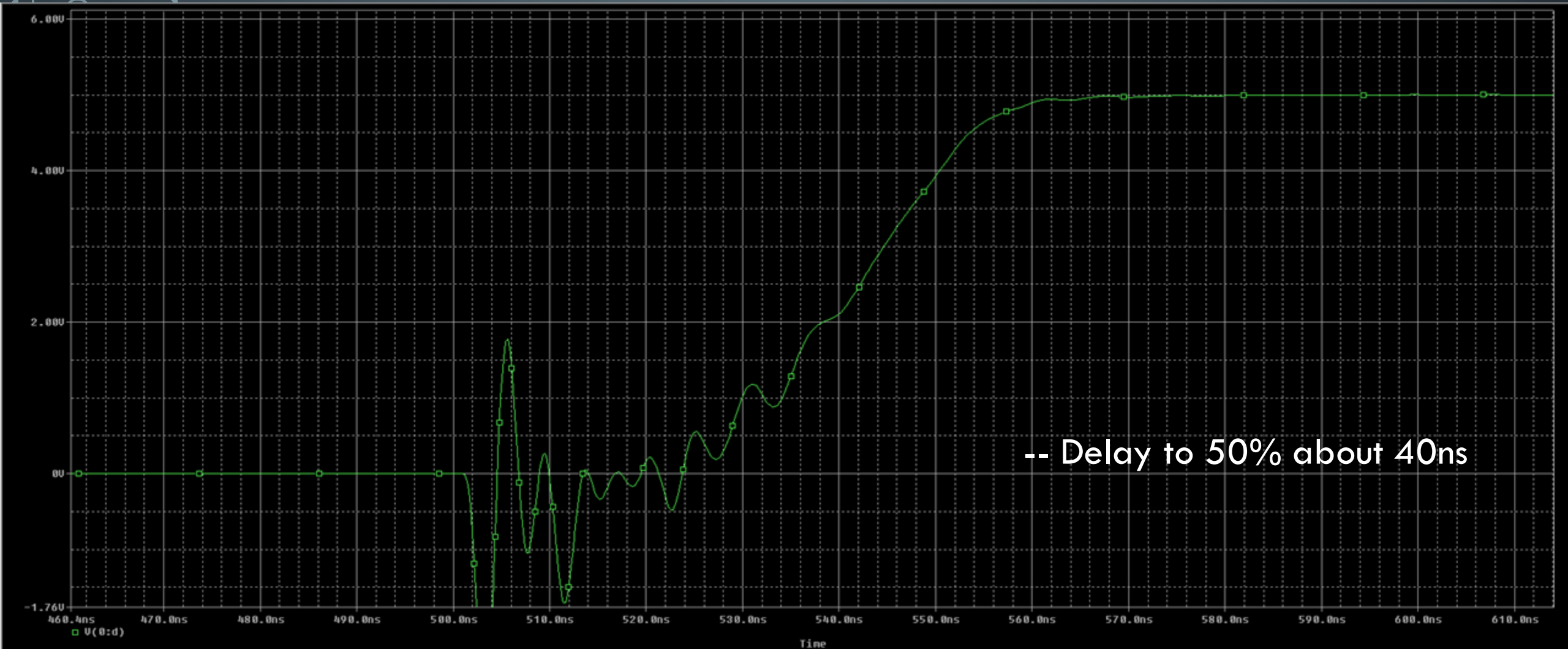
- Either A or B high will output high
- Both A and B high will output low
- Both A and B low will output low

PSPICE CMOS CIRCUIT DESIGN

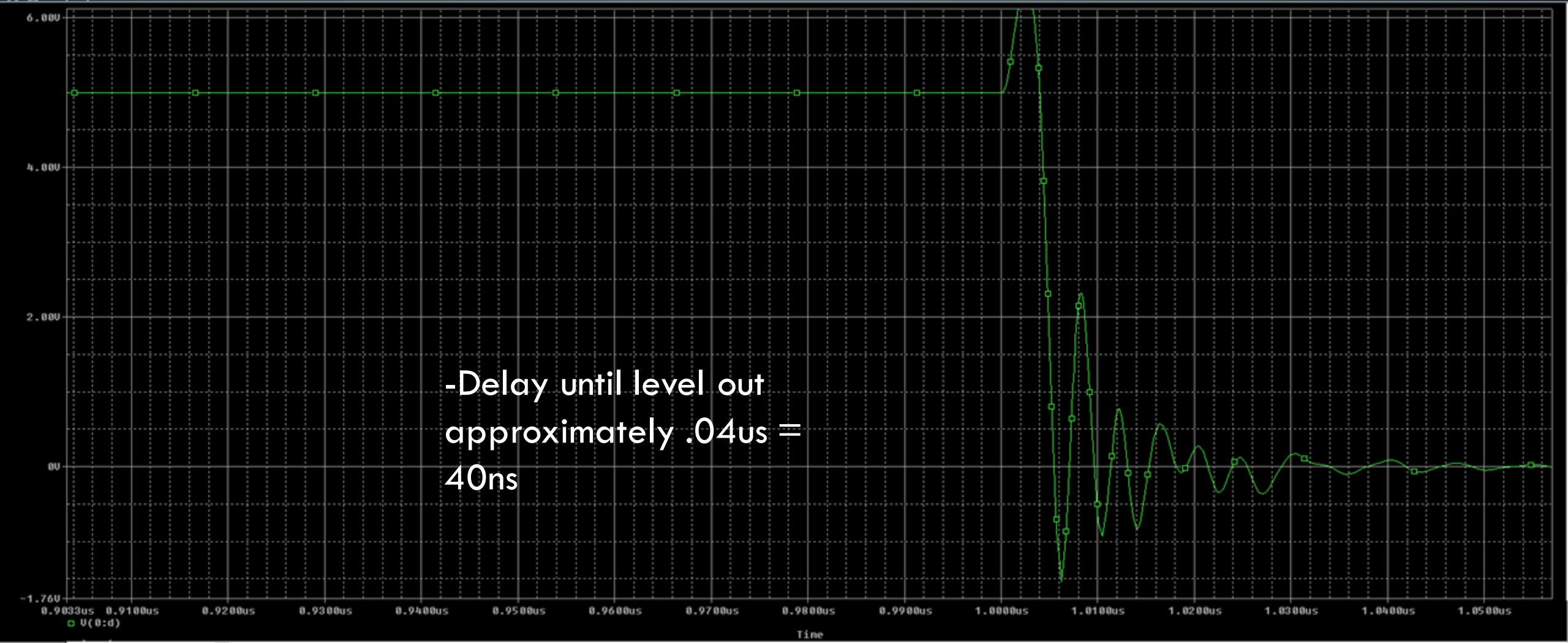




SIMULATION RESULTS (ACTIVE PULL UP - RISING EDGE)

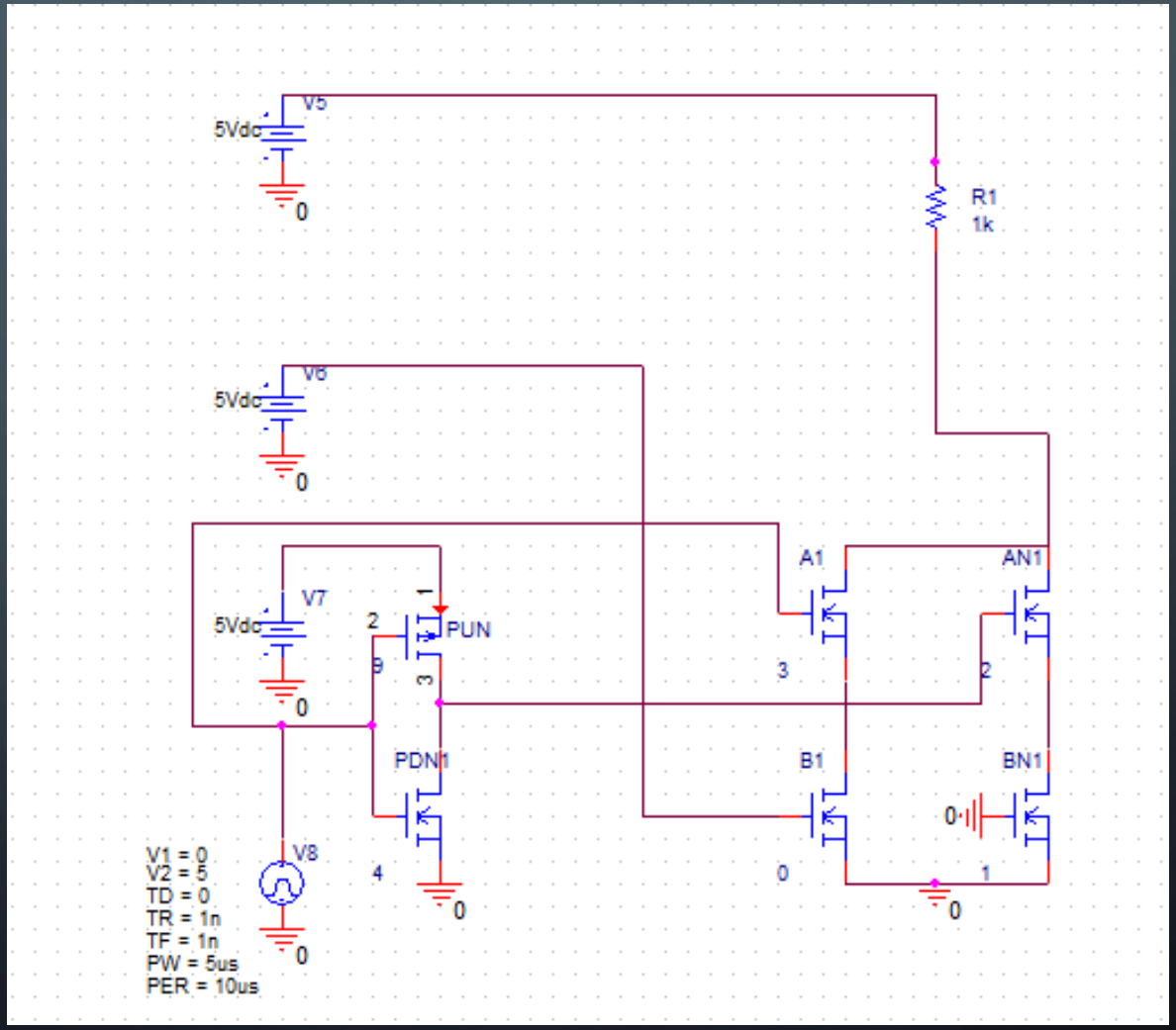


SIMULATION RESULTS (ACTIVE PULL UP - FALLING EDGE)

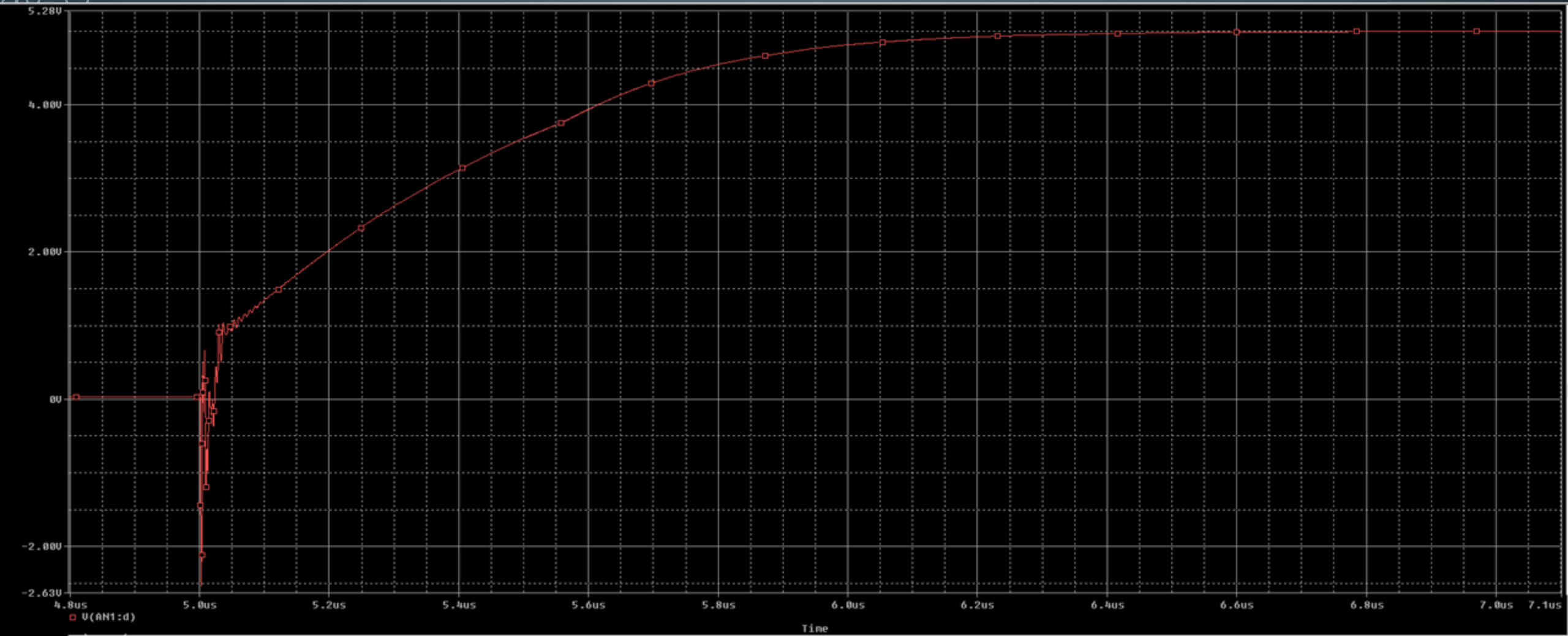


-Delay until level out approximately .04us = 40ns

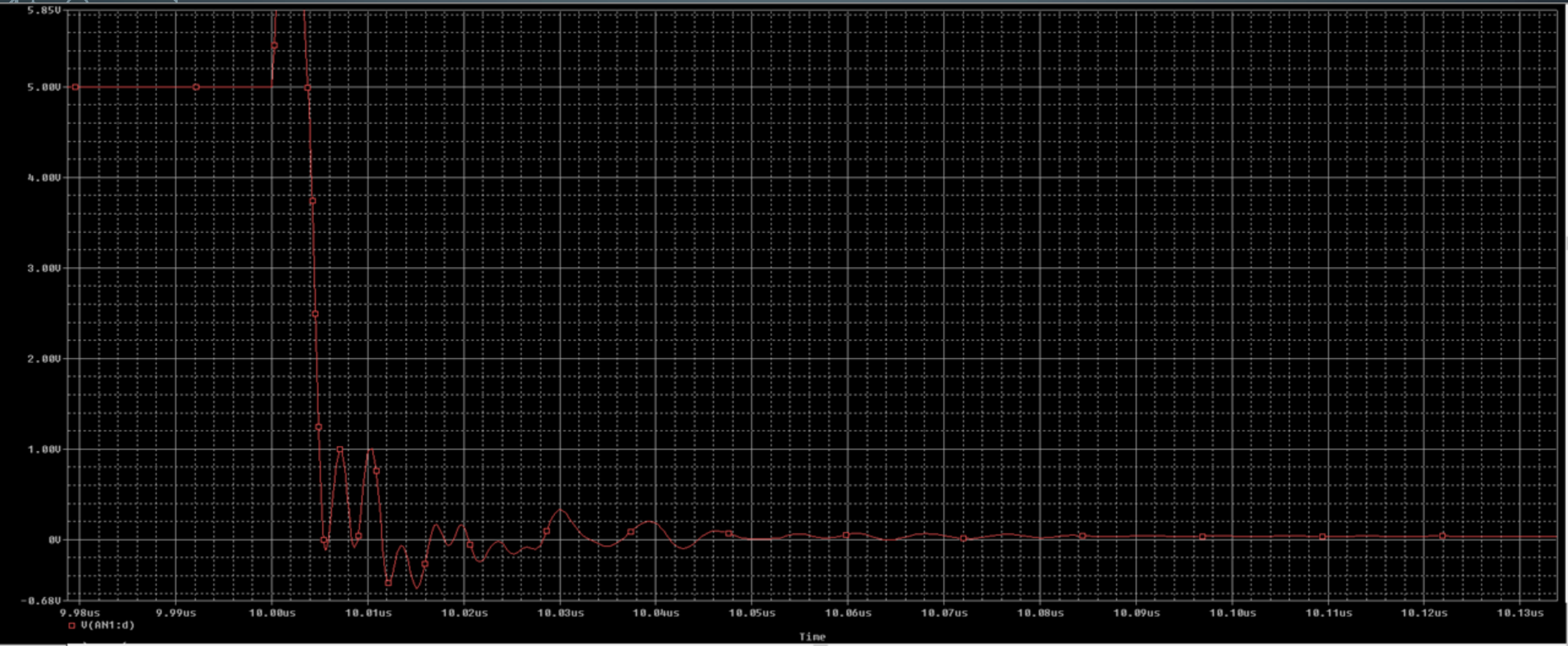
PSPICE RTL CIRCUIT DESIGN



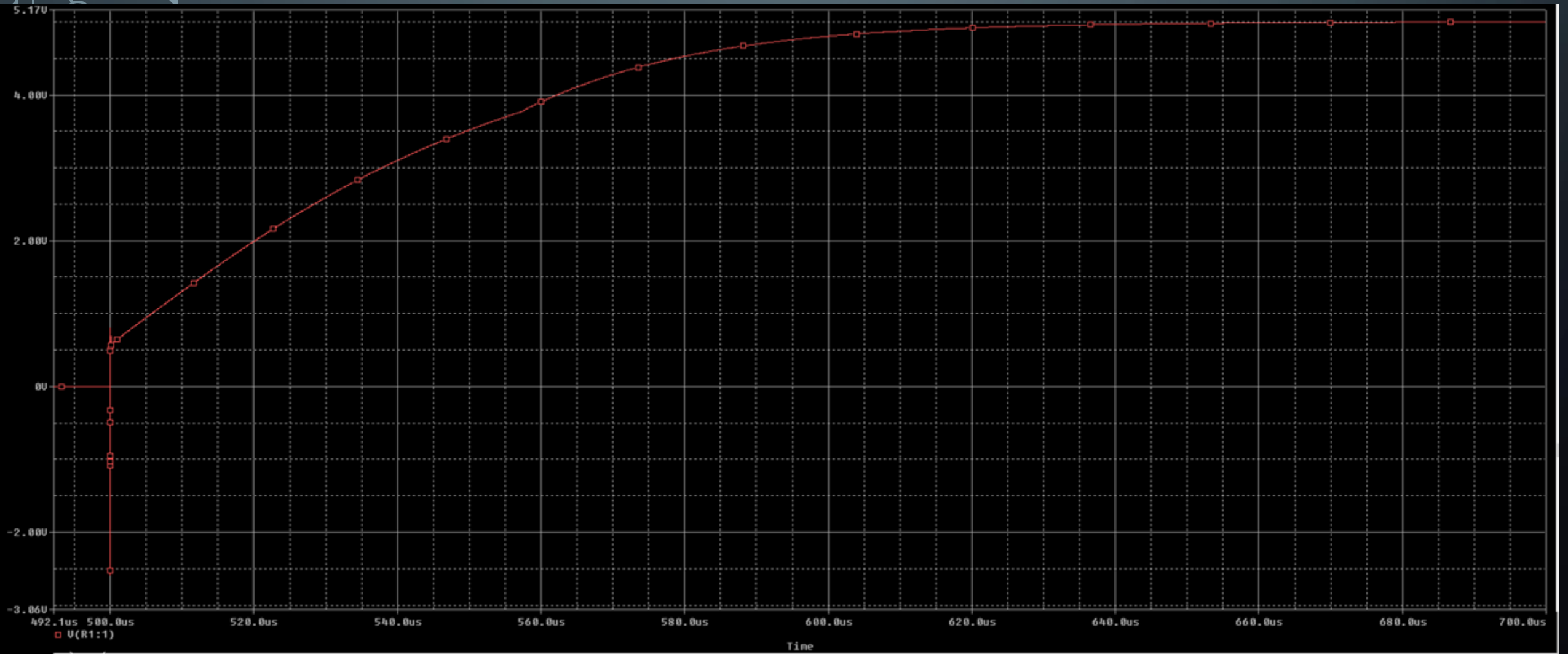
SIMULATION RESULTS (RESISTIVE PULL UP RISING EDGE 1K)



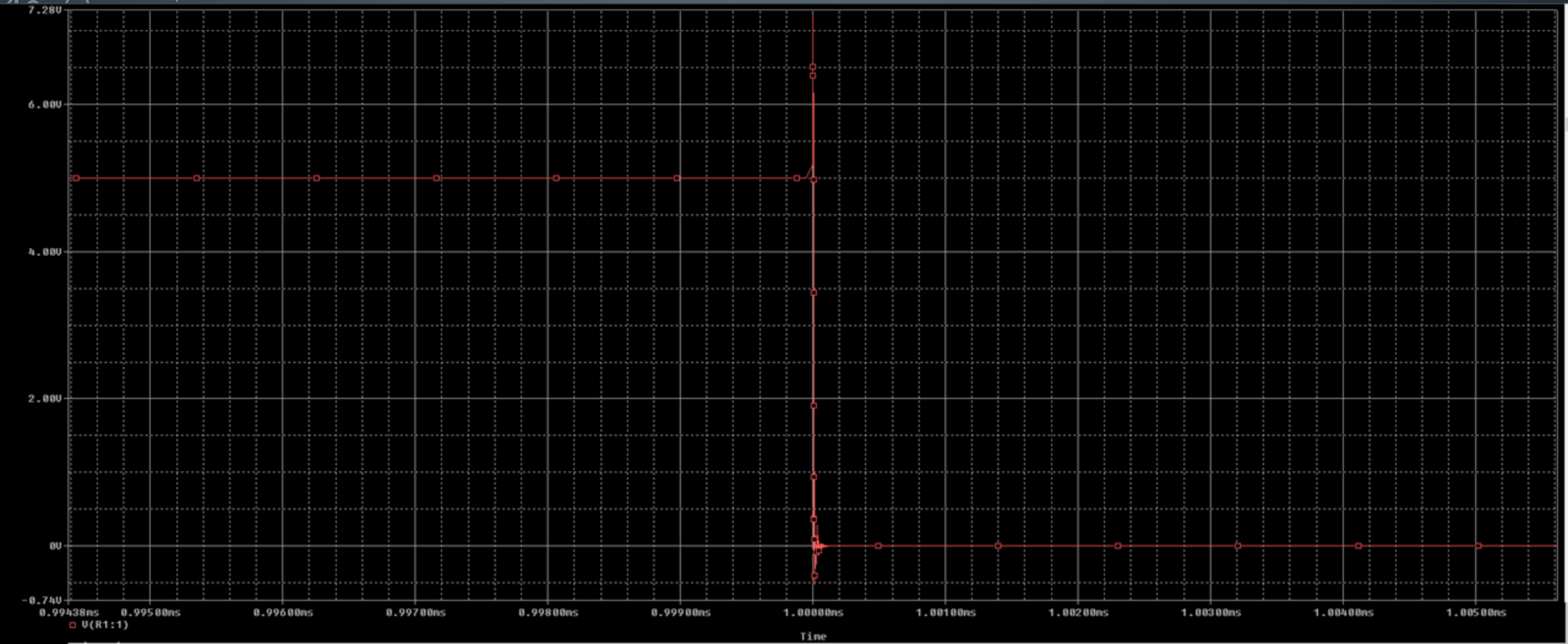
SIMULATION RESULTS (RESISTIVE PULL UP FALLING EDGE 1K)



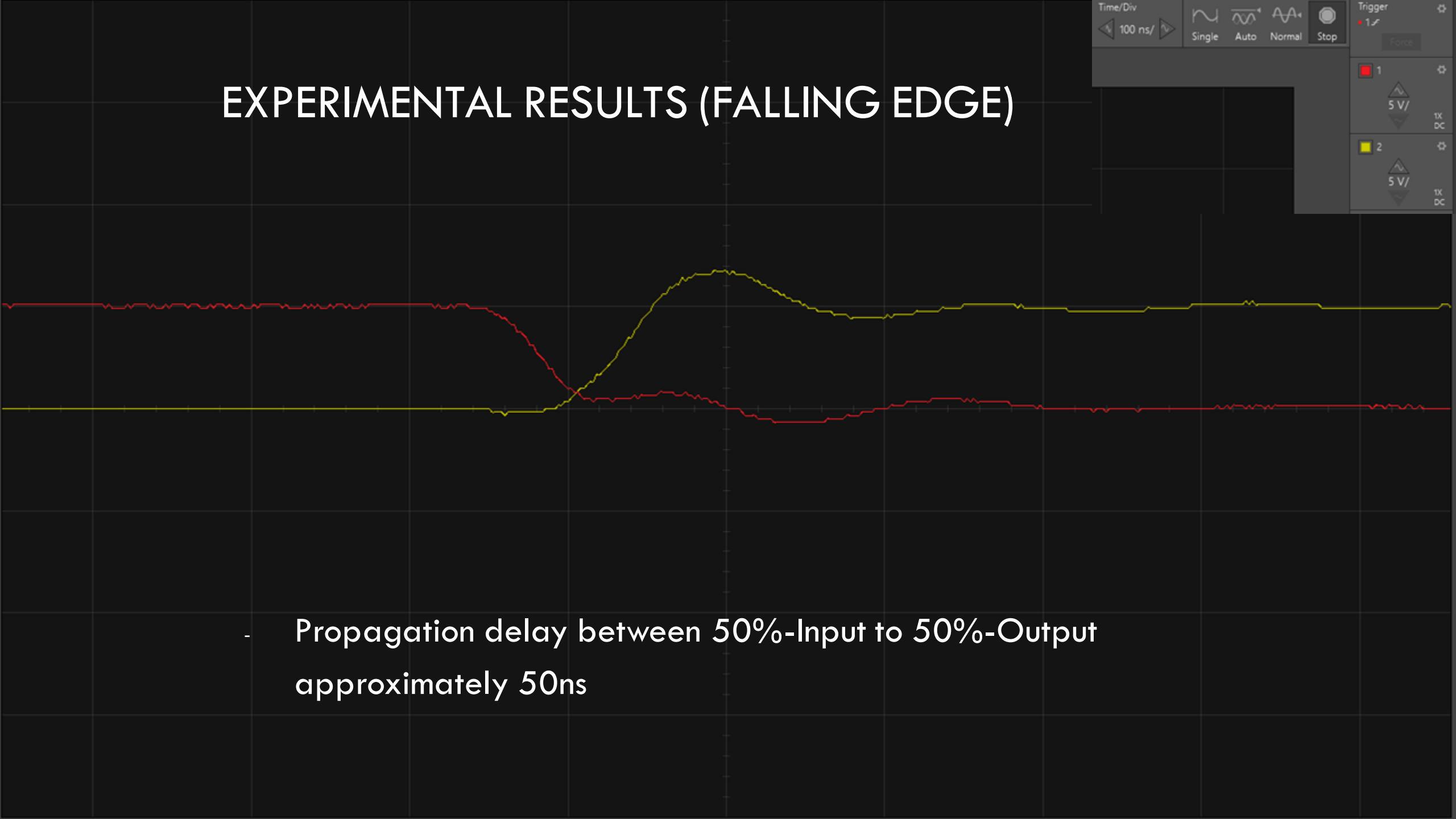
SIMULATION RESULTS (RESISTIVE PULL UP RISING EDGE 100K)



SIMULATION RESULTS (RESISTIVE PULL UP FALLING EDGE 100K)

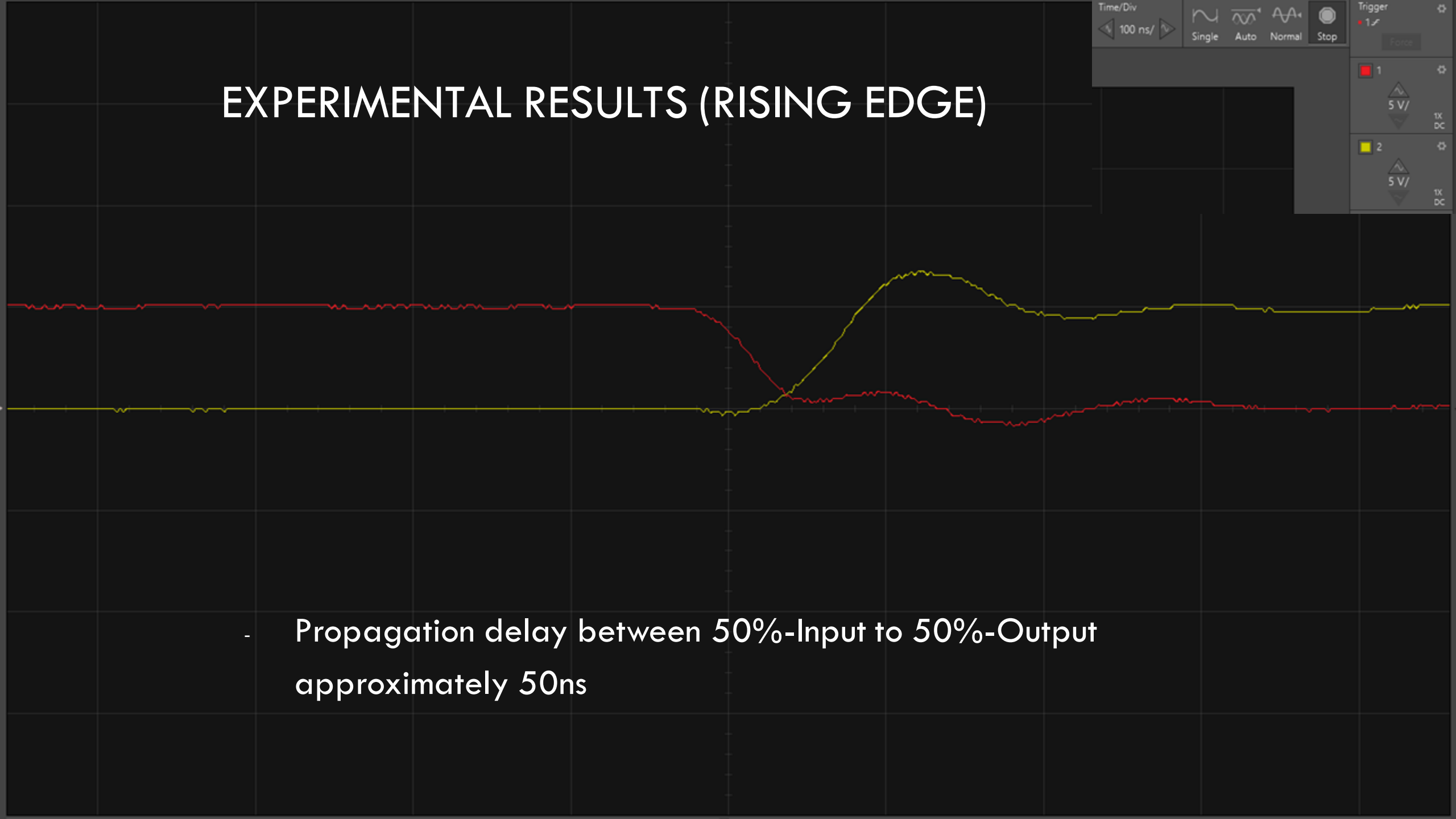


EXPERIMENTAL RESULTS (FALLING EDGE)



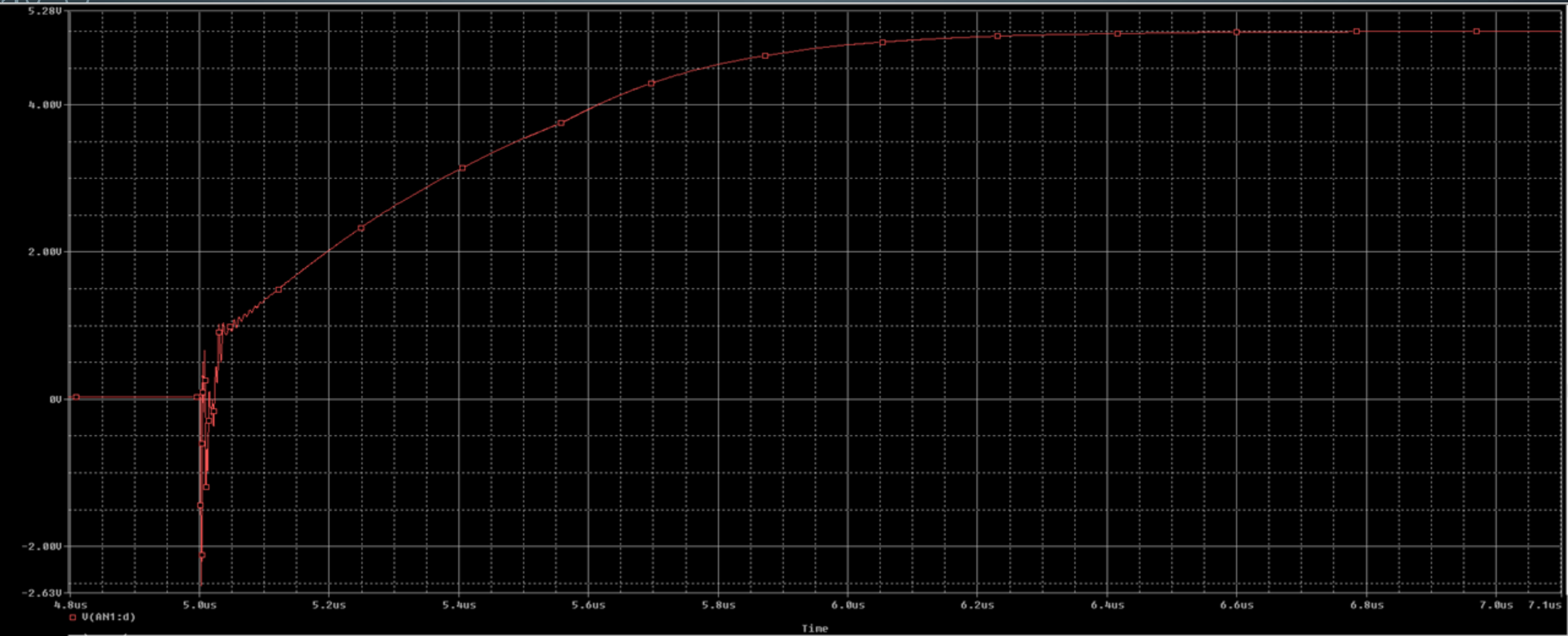
- Propagation delay between 50%-Input to 50%-Output approximately 50ns

EXPERIMENTAL RESULTS (RISING EDGE)

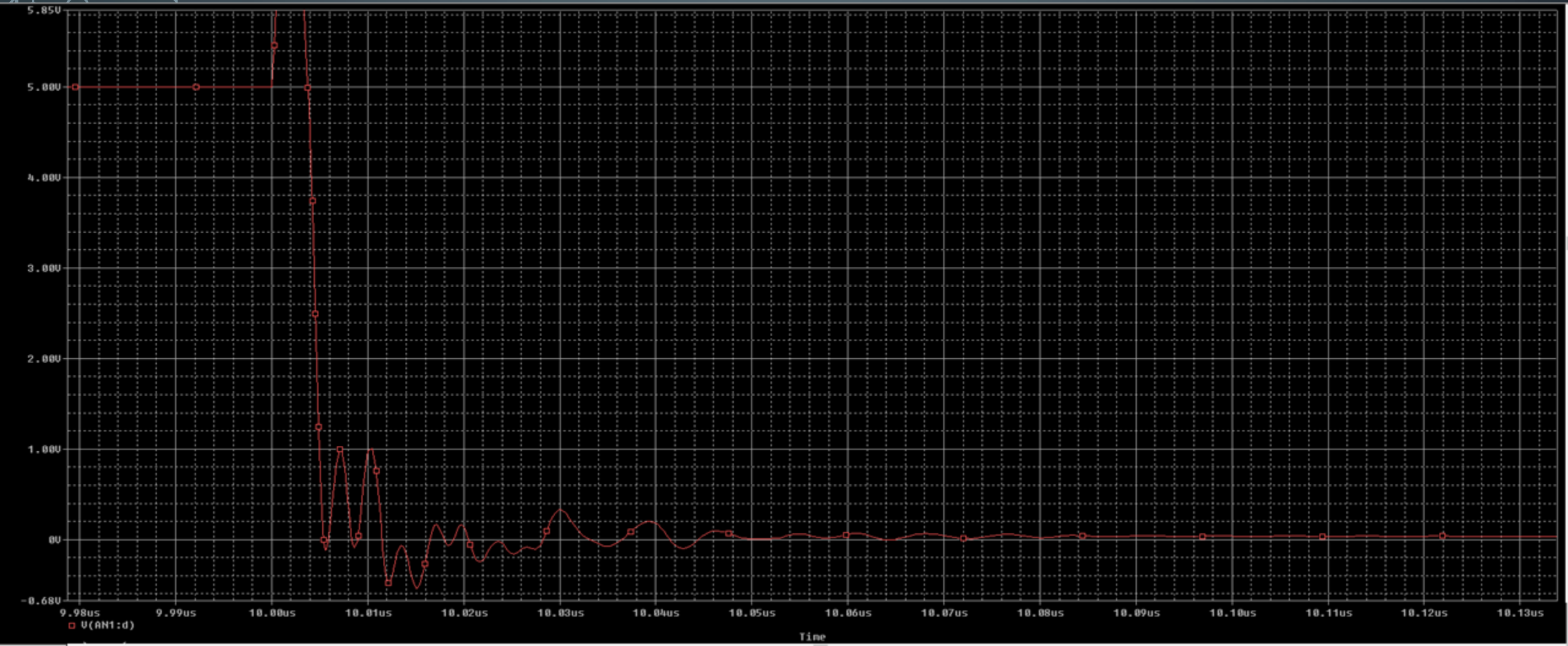


- Propagation delay between 50%-Input to 50%-Output approximately 50ns

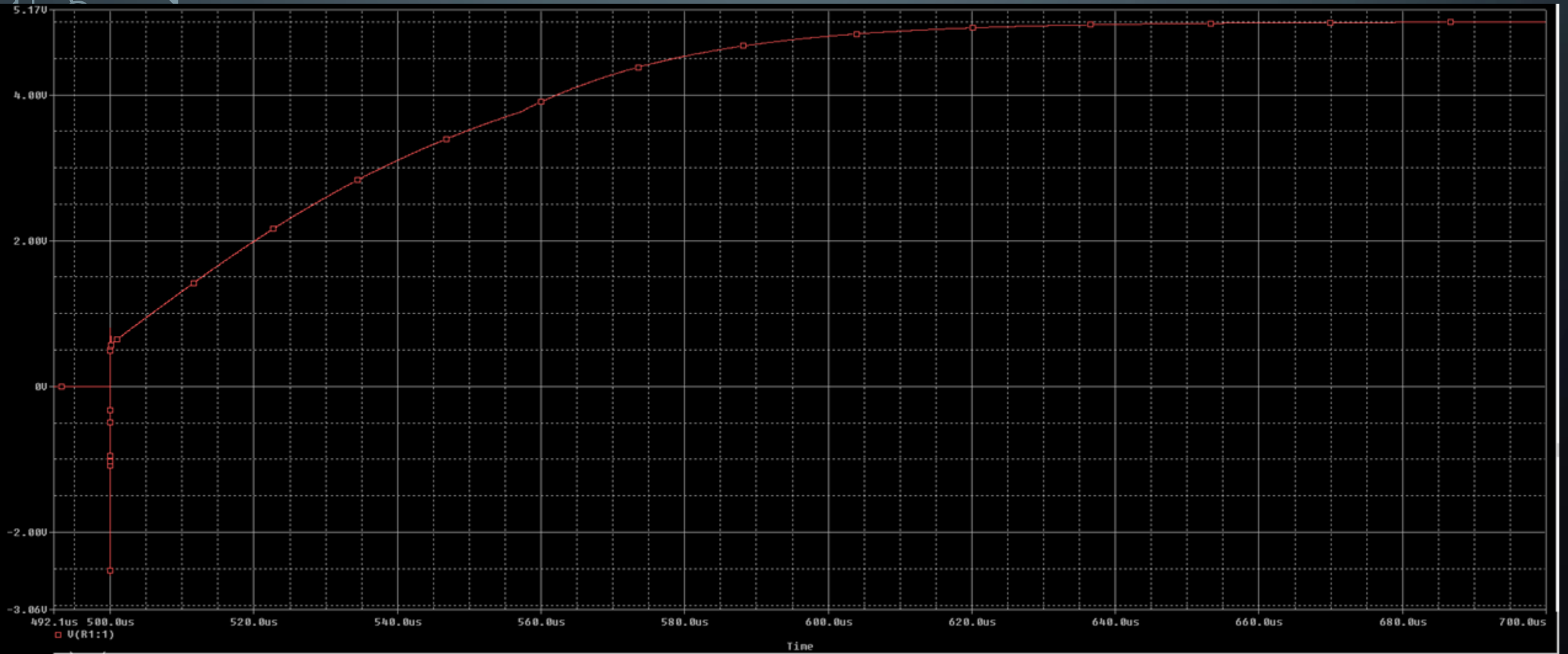
SIMULATION RESULTS (RESISTIVE PULL UP RISING EDGE 1K)



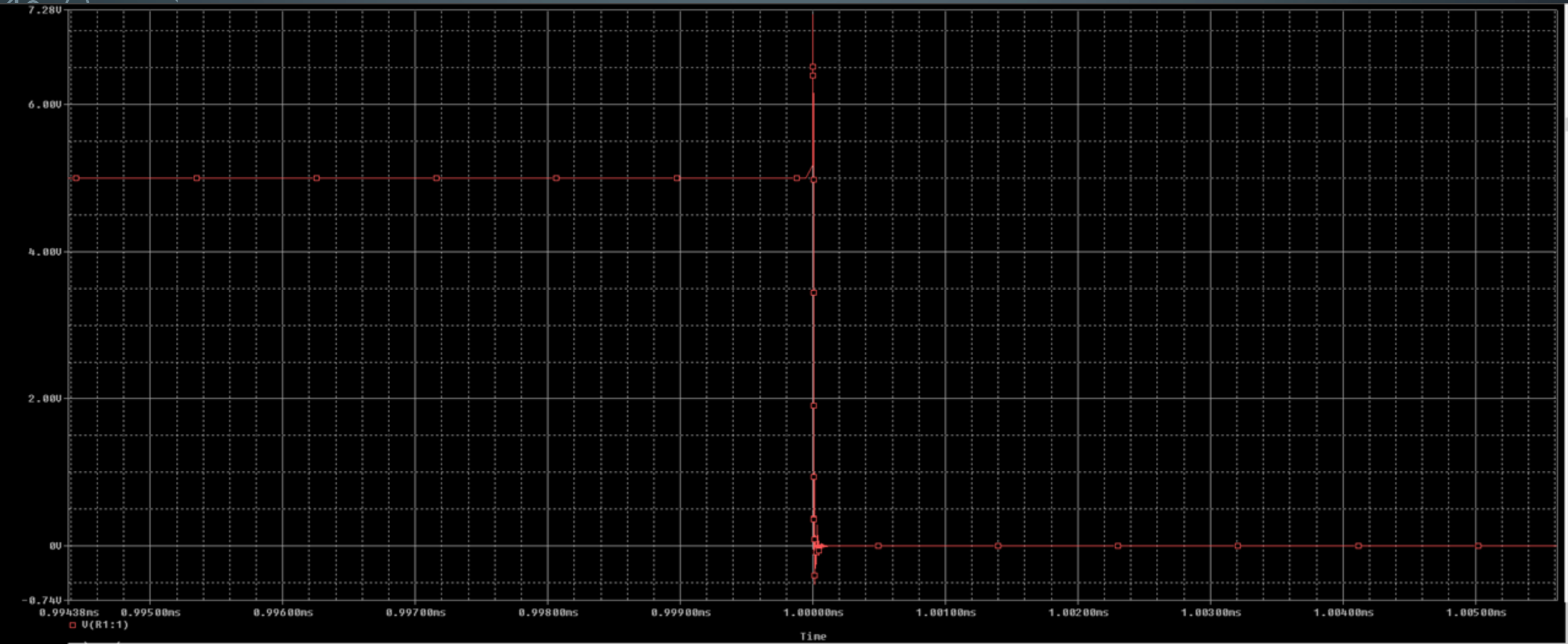
SIMULATION RESULTS (RESISTIVE PULL UP FALLING EDGE 1K)



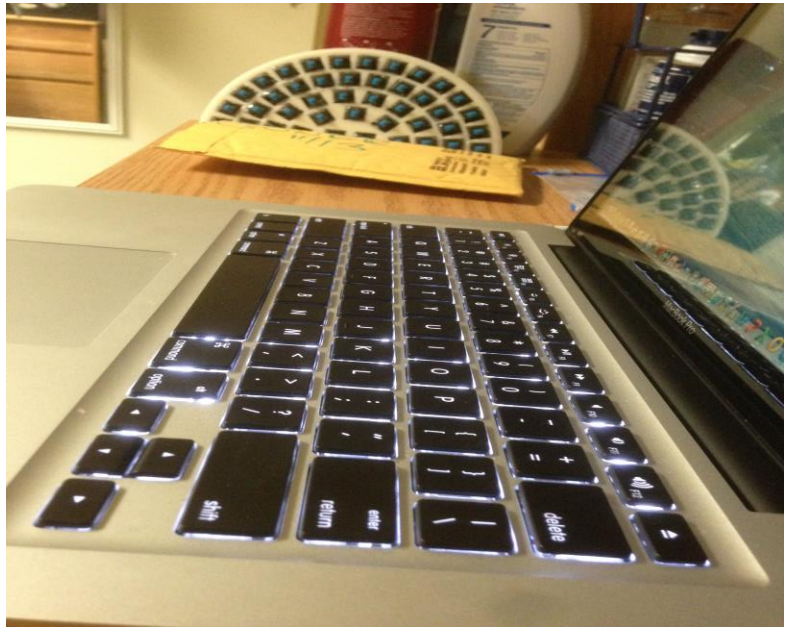
SIMULATION RESULTS (RESISTIVE PULL UP RISING EDGE 100K)



SIMULATION RESULTS (RESISTIVE PULL UP FALLING EDGE 100K)



ENG 142: FUNDAMENTALS OF ENGINEERING: TERM PROJECT (AWARDED INNOVATIVE DESIGN 2015)



Casing designed in SolidWorks and fabricated via TCNJ's 3D printer

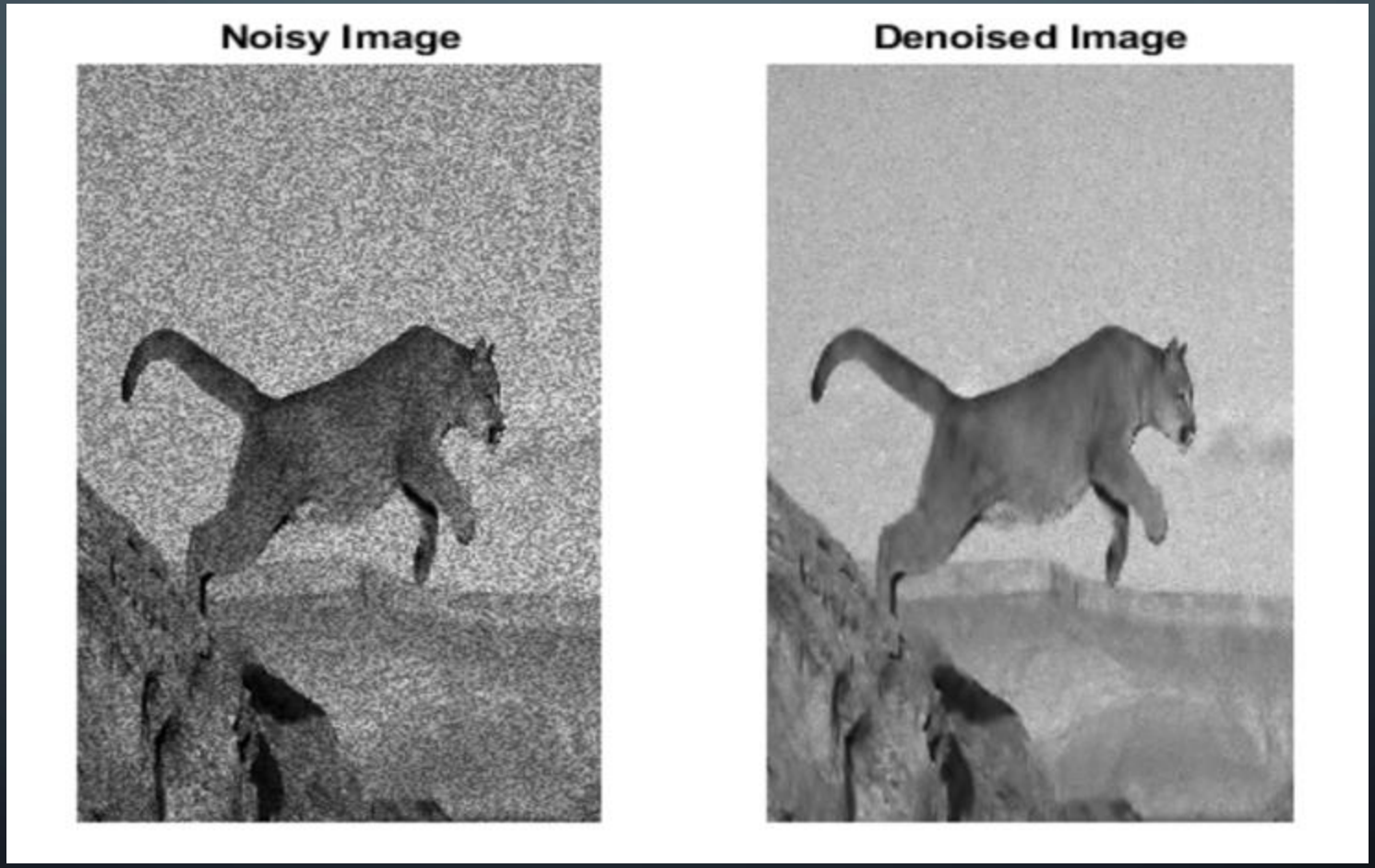
Ergonomic placement of characters decided by result of character frequency counter (written in Java) as a substitute for the conventional qwerty layout.

ONE HANDED KEYBOARD

SIGNAL DENOISING USING WAVELET TRANSFORMATION

BRYAN GUNER

IMAGE DENOISING



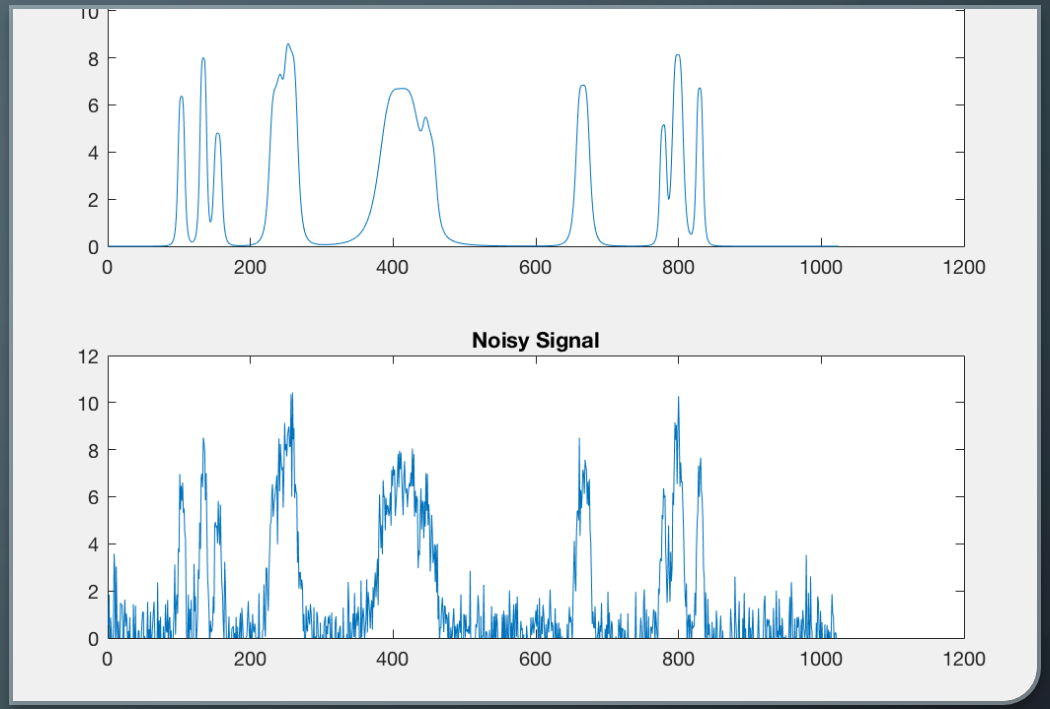
CONCEPT

- Compared to other denoising techniques, wavelet and wavelet packet denoising allows the user to retain features in the data
- Data can be compressed by by setting seemingly unimportant wavelet/wavelet packets coefficients to zero and reconstructing data
- Interval-dependent thresholds can be applied to denoise data with non-constant variance since signal noise is not always uniform in time

METHODS

- Localize features in your data to different scales
- You can preserve important signal or image features while removing noise
- The wavelet transform leads to a sparse representation for many real-world signals and images
- The wavelet transform concentrates signal and image features in a few large-magnitude wavelet coefficients
- Wavelet coefficients which are small in value are typically noise and you can "shrink" those coefficients or remove them without affecting the signal or image quality. After you threshold the coefficients, you reconstruct the data using the inverse wavelet transform

```
rng default;
[X,XN] = wnoise('bumps',10,sqrt(6
subplot(211)
plot(X); title('Original Signal')
AX = gca;
AX.YLim = [0 12];
subplot(212)
plot(XN); title('Noisy Signal');
AX = gca;
AX.YLim = [0 12];
```

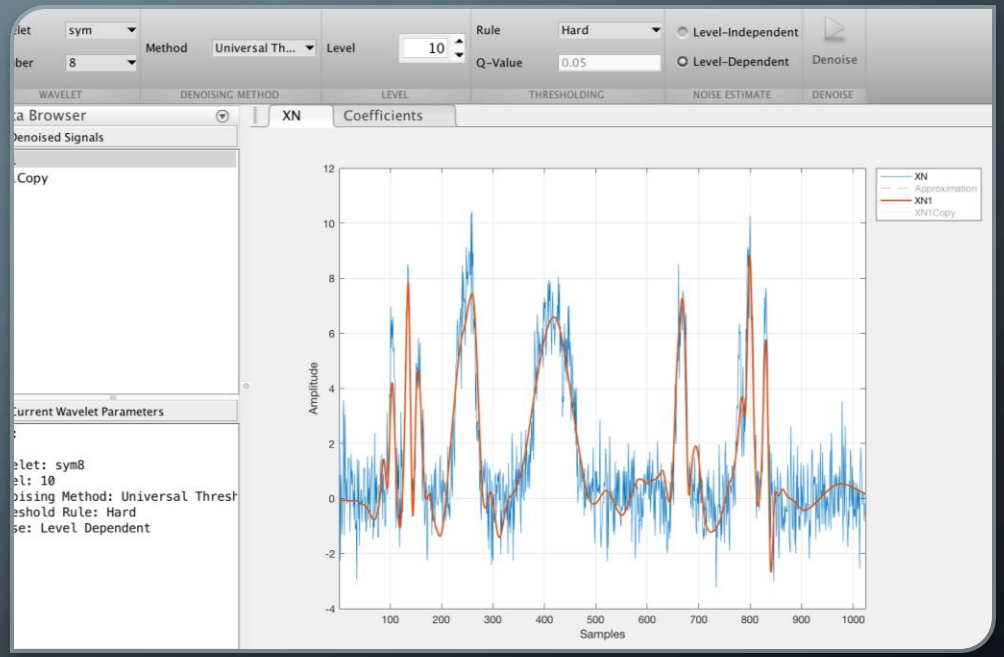


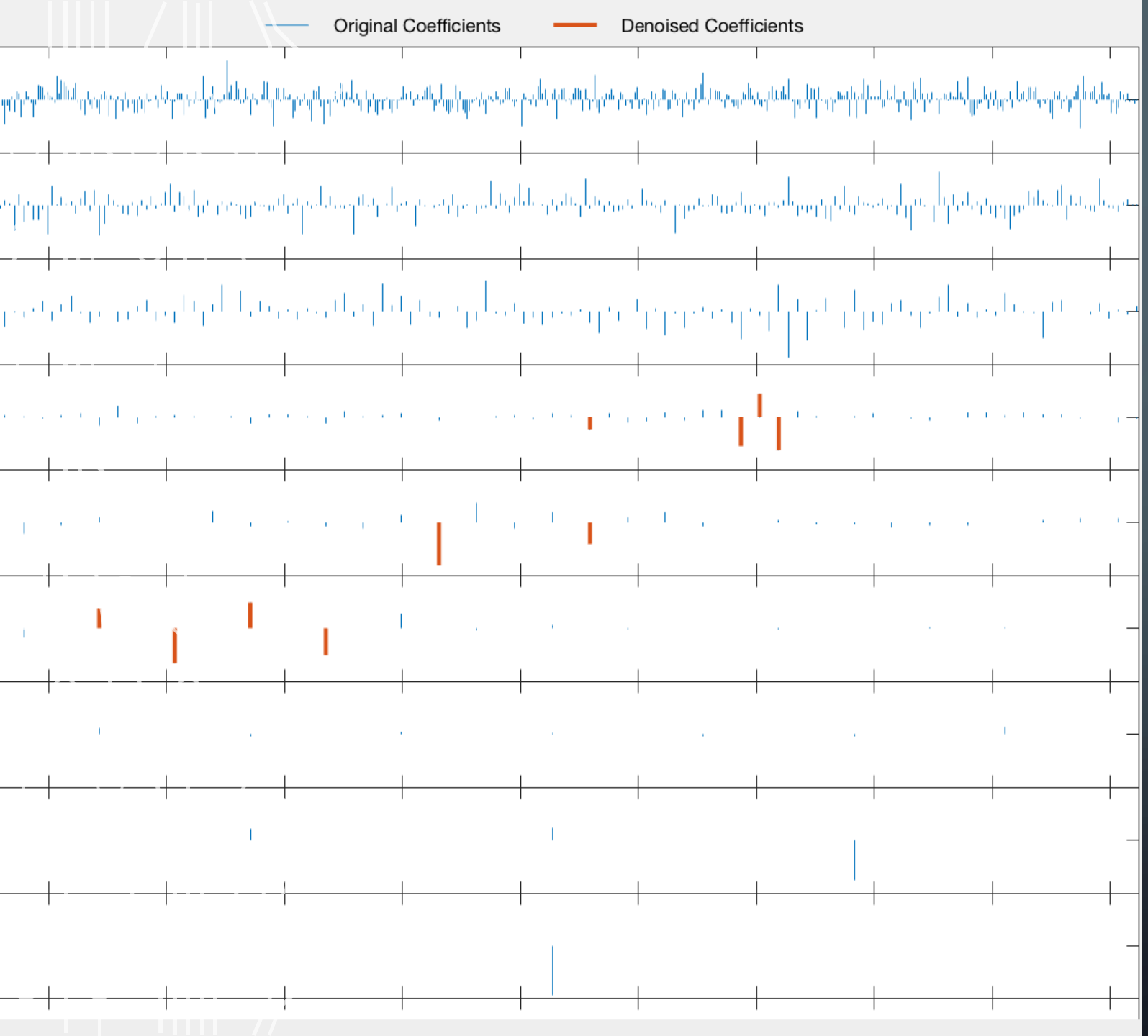
MATLAB IMPLEMENTATION

NOISY SIGNAL SHOWS ORIGINAL SIGNAL WITH ADDED WHITE NOISE

MATLAB WAVELET DENOISING METHOD 1

- Symlet Wavelet Family (Symmetrical)
- Universal Threshold

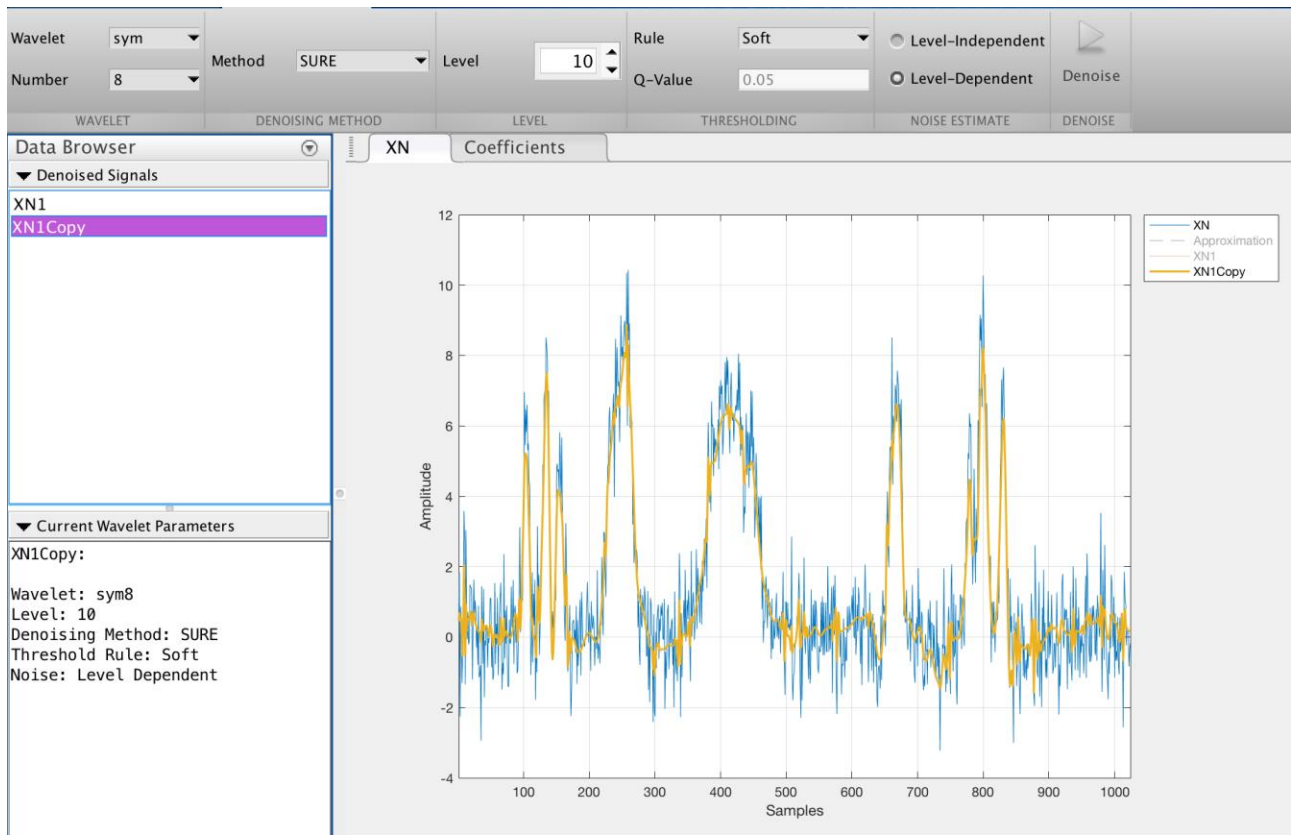




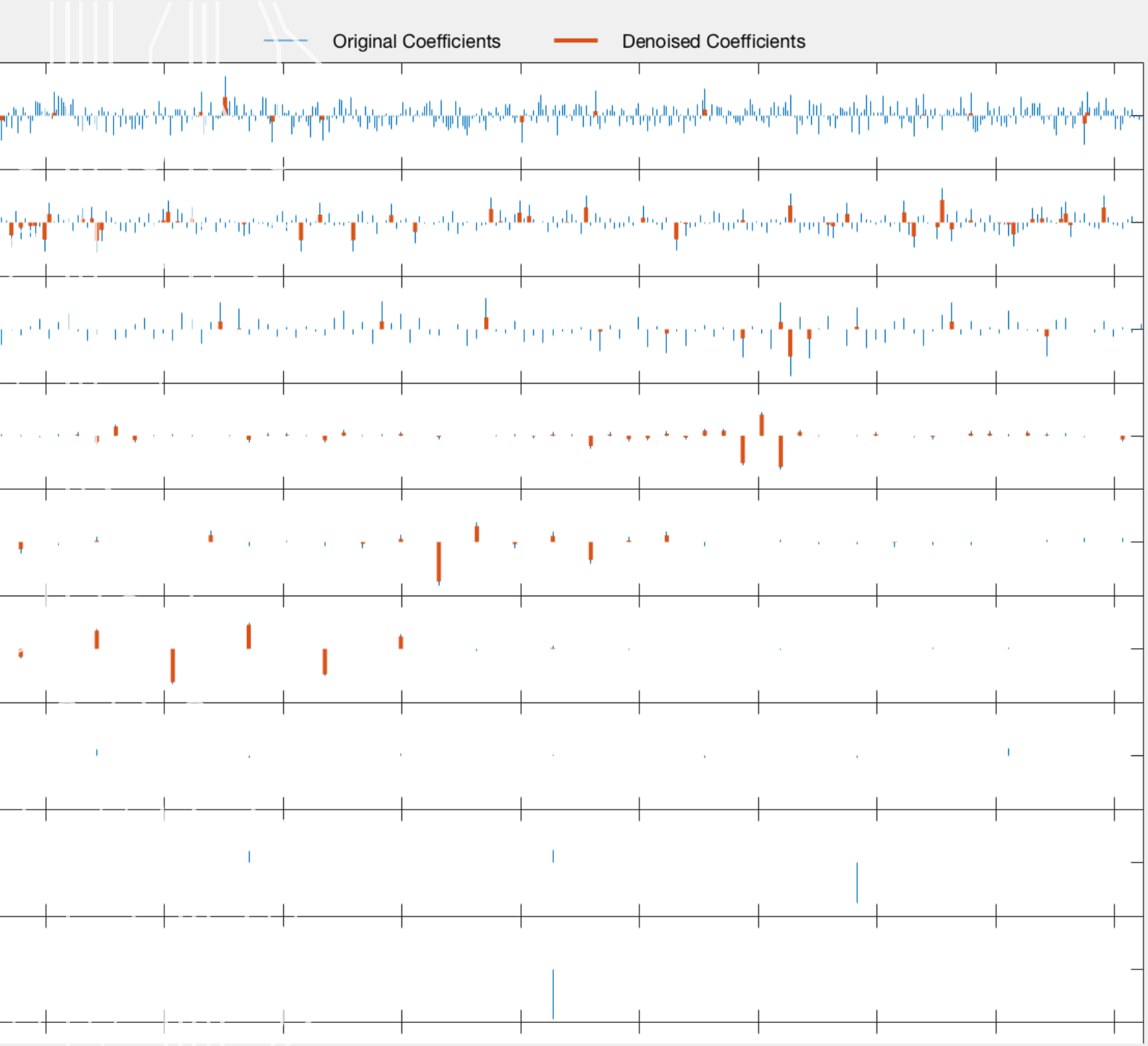
COEFFICIENT GRAPH

- Original coefficients shown in blue
- Denoised coefficients shown in orange
- Samples(X) Vs. Level(Y)

MATLAB WAVELET DENOISING METHOD 2

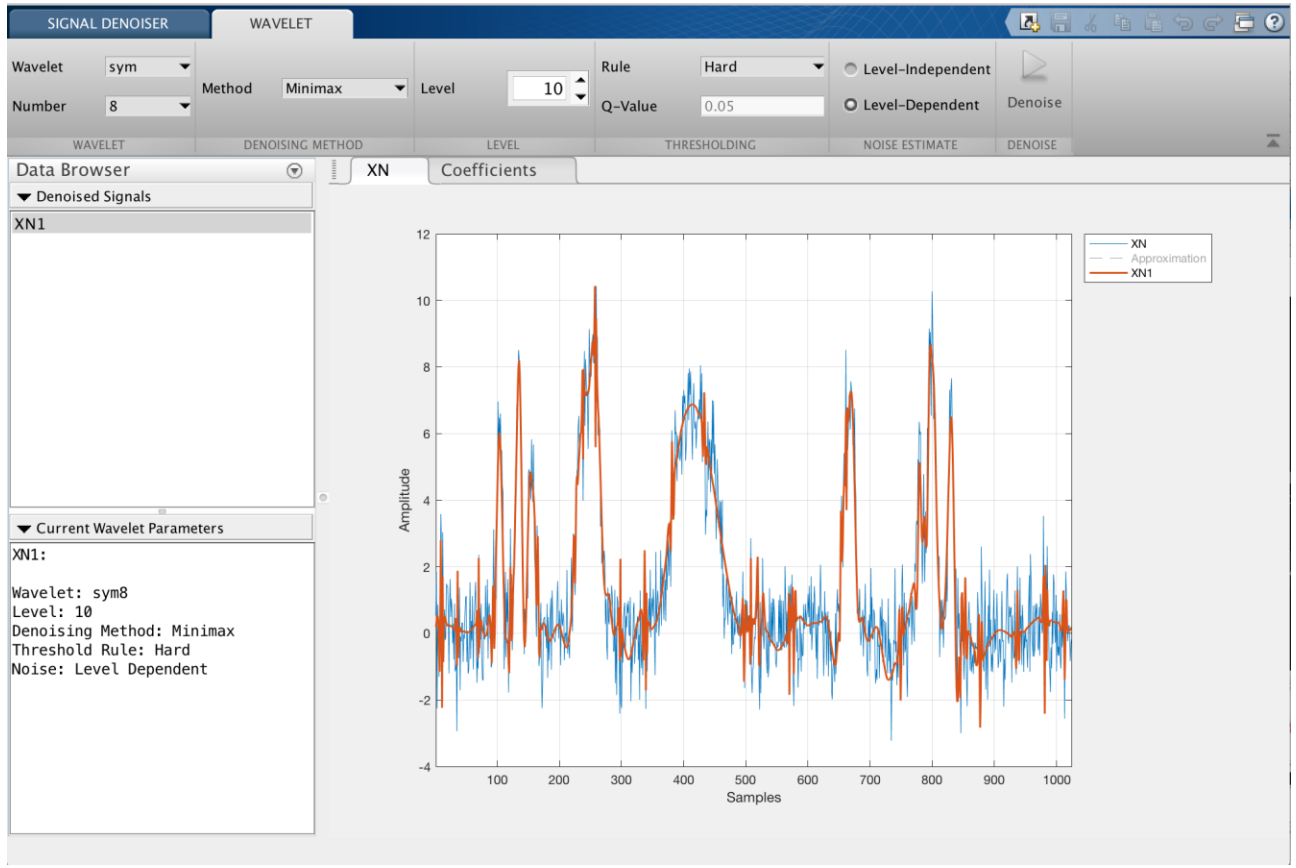


- Symlet Wavelet Family (Symmetrical)
- SURE Method



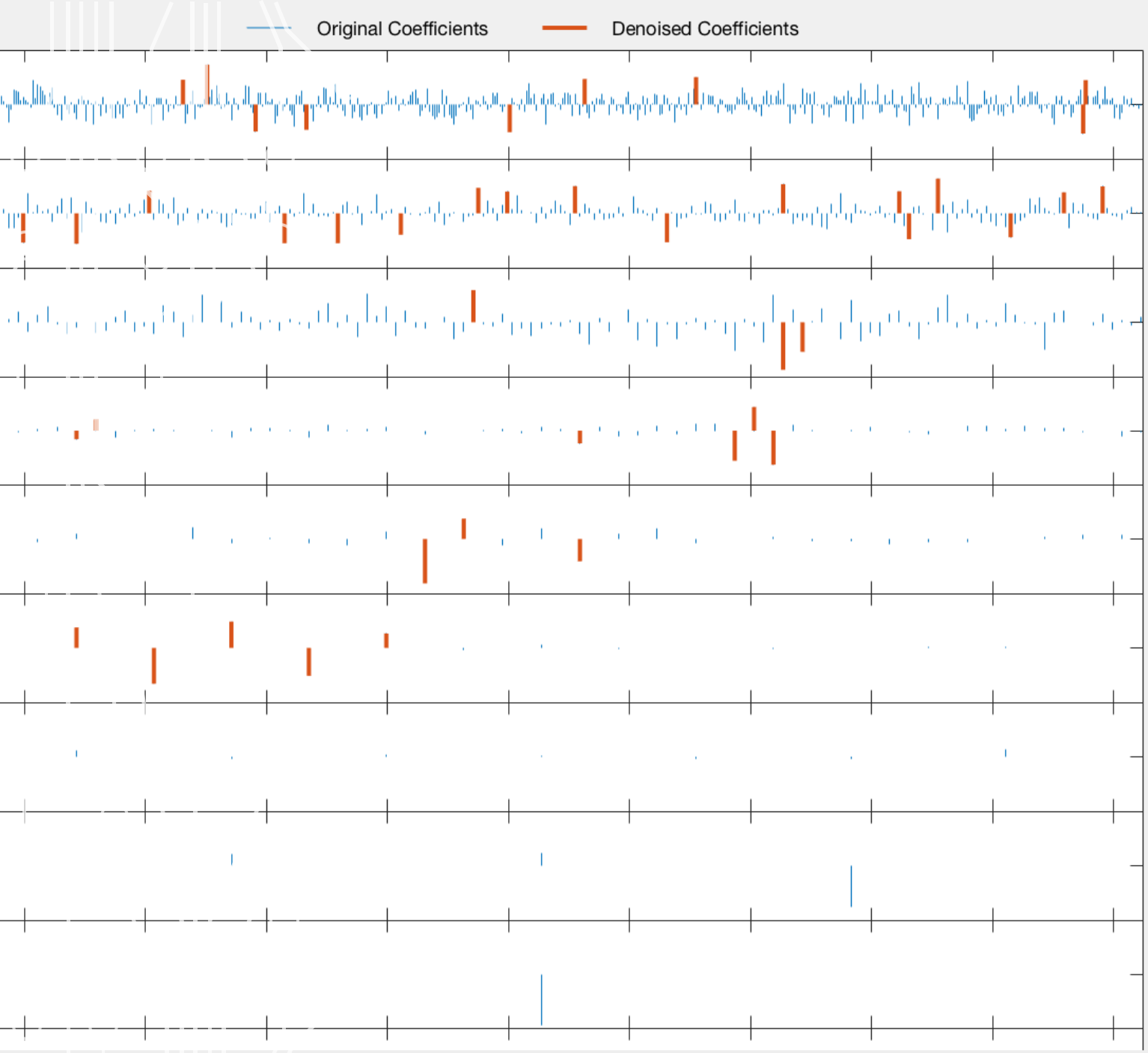
COEFFICIENT GRAPH

- Original coefficients shown in blue
- Denoised coefficients shown in orange
- Samples(X) Vs. Level(Y)



MATLAB WAVELET DENOISING METHOD 3

- Symlet Wavelet Family (Symmetrical)
- Minimax Method



COEFFICIENT GRAPH

- Original coefficients shown in blue
- Denoised coefficients shown in orange
- Samples(X) Vs. Level(Y)