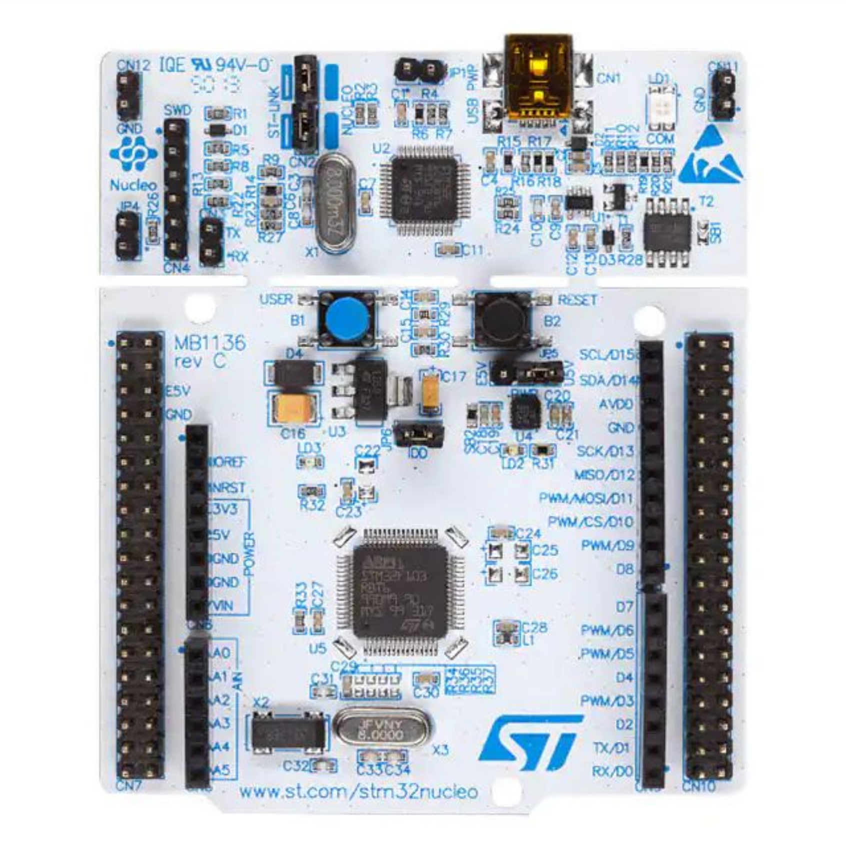
DSP Final Lab

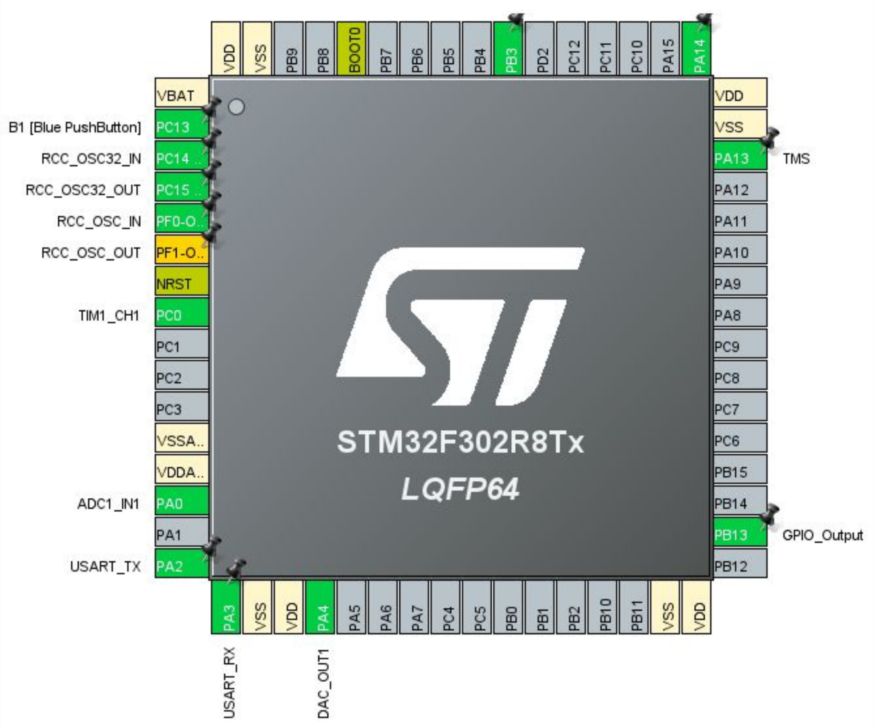
Matthew Bohr

Objectives

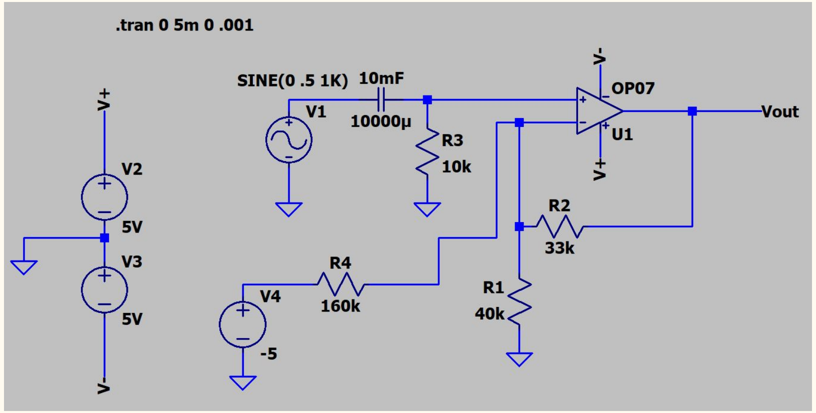
- Implement notch filter using C.
- Create signal conditioning circuit.
- Implement digital notch filter on microcontroller.
- Process audio signal using digital filter.

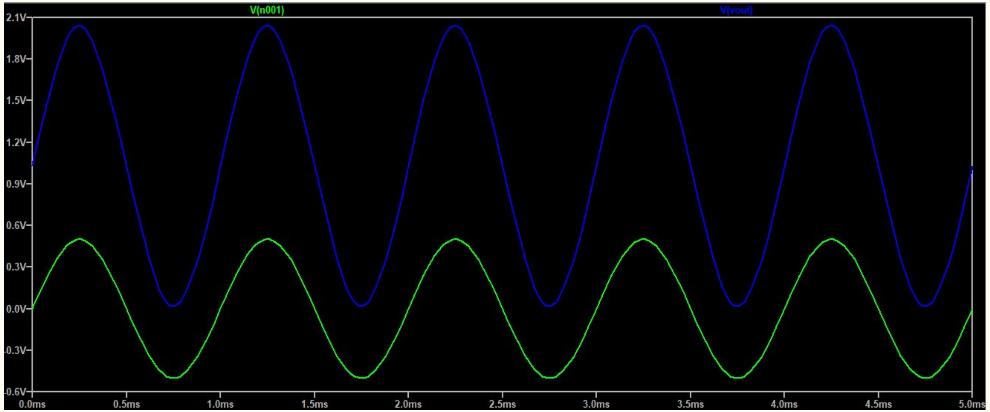
STM32-F302R8 Nucleo Board

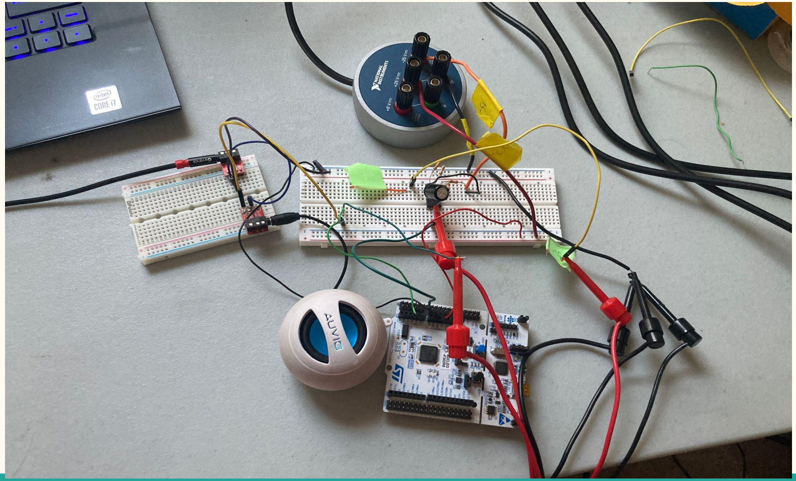


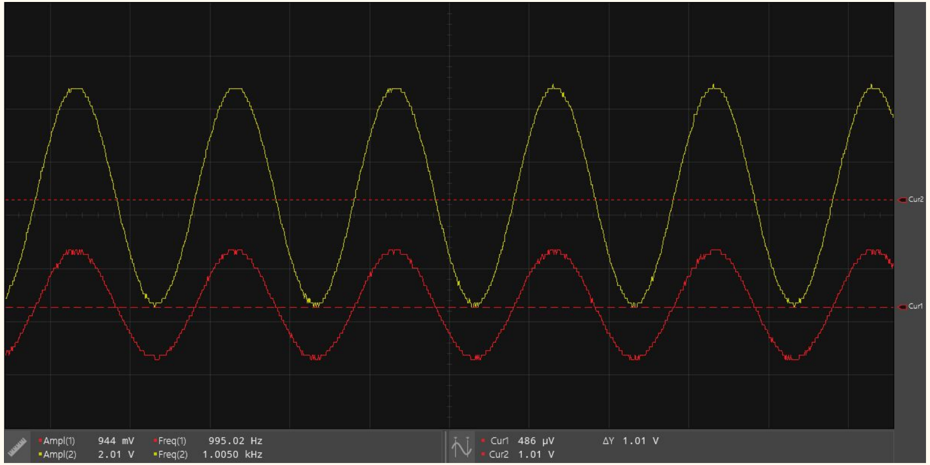


Signal Conditioning Circuit



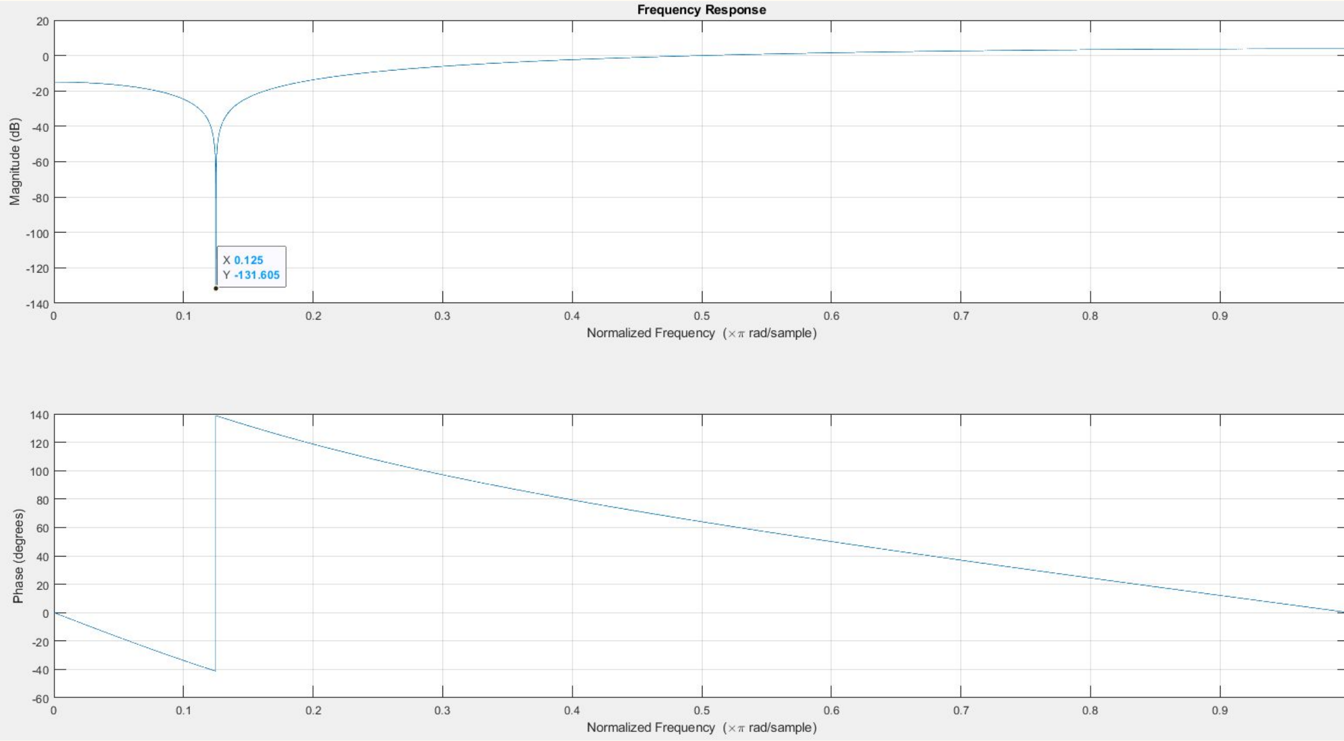






Timer Frequency, changes on both edges

Notch Filter Frequency Response




```
clear
close all

F = 32000;
alpha = 0.9;
filter_omega = 3.14159/8;

a = [2*alpha*cos(filter_omega), -1*alpha^2];
b = [1, -2 * cos(filter_omega), 1];

freqz(b,a,F); title('Frequency Response')
```

$$w_0 = \frac{2\pi f_0}{F_t} \quad \Longrightarrow \quad f_0 = \frac{w_0 F_t}{2\pi} = \frac{\frac{\pi}{8} 32000}{2\pi} = 2000Hz$$

C Code Filter Implementation

1. Filter Structure
2. Filter Function
3. Filter Parameters
4. Creating Filter
5. Filtering Input

```
#include <iostream>
#include <fstream>
#include <string>
#include <math.h>

// struct to hold filter states and coeffs
typedef struct
{
    double a1, a2;
    double b0, b1, b2;
    double z1,z2;
}biquad_dbl_t;

// filter prototype, accepts one input and produces one output
double biquad_dbl(double xm, biquad_dbl_t* bq)
{
    double ym = bq->b0*xm + bq->z1;
    bq->z1 = bq->b1*xm + bq->a1*ym + bq->z2;
    bq->z2 = bq->b2*xm + bq->a2*ym;
    return ym;
}
int main()
{
    FILE *fp;
    fp = fopen("out3.csv", "w");            // writing to out.csv
    const int f0 = 3000;                     // cosine frequency
    const int Fs = 32000;                    // sampling frequency
    double alpha = 0.90;

    const double PI = 3.14159;
    const double filter_omega = PI/8;        // notch filter frequency (=2000Hz if Fs = 32000Hz)
    const int sig_len = 200;                 // num samples

    double x[sig_len];                       // input vector
    double y[sig_len];                       // vector to hold filter output

    // create filter and compute coeffs
    biquad_dbl_t filter1;
    filter1.a1 = 2*alpha*cos(filter_omega);
    filter1.a2 = -1*pow(alpha,2);
    filter1.b0 = 1;
    filter1.b1 = -2 * cos(filter_omega);
    filter1.b2 = 1;

    // computing output sequence
    for (int i=0; i<sig_len-1; i++)
    {
        x[i] = cos(2*PI*((double)f0/Fs)*i);
        y[i] = biquad_dbl(x[i], &filter1);
        fprintf(fp, "%10.8f,%f\n", x[i], y[i]);
    }
    fclose(fp);
}
```

STM32 Filter Struct, Function, and ISR

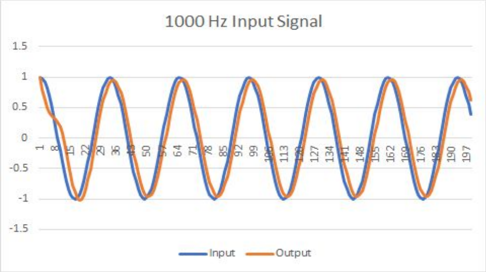
```c
typedef struct
{
    double a1, a2;
    double b0, b1, b2;
    double z1, z2;
}biquad_dbl_t;

double biquad_dbl(double xm, biquad_dbl_t* bq)
{
    double ym = bq->b0*xm + bq->z1;
    bq->z1 = bq->b1*xm + bq->a1*ym + bq->z2;
    bq->z2 = bq->b2*xm + bq->a2*ym;
    return ym;
}
```

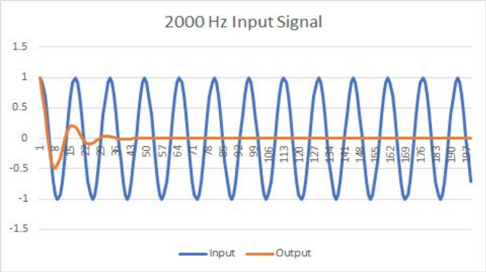
```c
double alpha = 0.90;
const double PI = 3.14159;
const double filter_omega = PI/8;

filter1.a1 = 2*alpha*cos(filter_omega);
filter1.a2 = -1*pow(alpha,2);
filter1.b0 = 1;
filter1.b1 = -2 * cos(filter_omega);
filter1.b2 = 1;
```

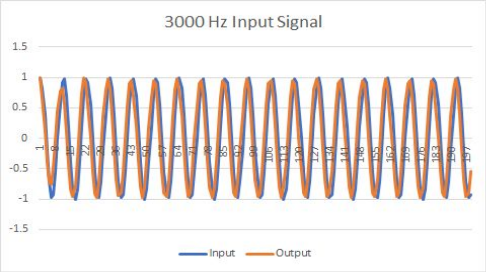
```c
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef *hadc)
{
  static double y;
  uint32_t adc_val;
  // storing ADC value in adc_val
  adc_val = HAL_ADC_GetValue(&hadc1);
  // processing adc_val through filter
  y  = biquad_dbl(adc_val, &filter1);
  // Toggle the Green LED
  HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_13);
  // writing to DAC
  HAL_DAC_SetValue(&hdac, DAC_CHANNEL_1, DAC_ALIGN_12B_R, ((uint32_t)y));
  HAL_DAC_Start(&hdac, DAC_CHANNEL_1);
}
```

C Code Results vs STM32 Filter Results

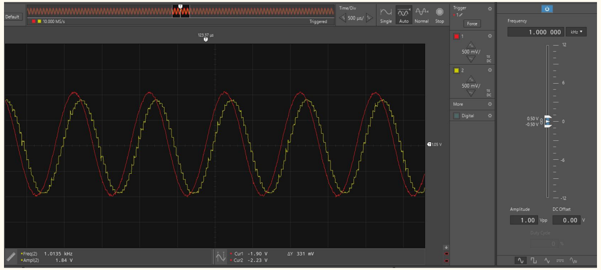
C Code Results

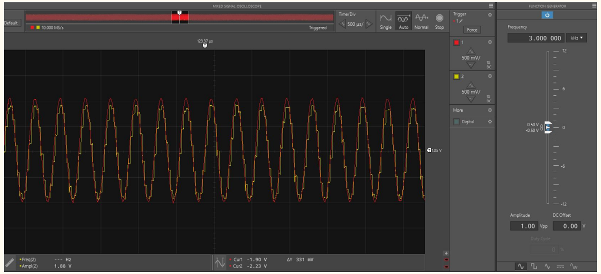




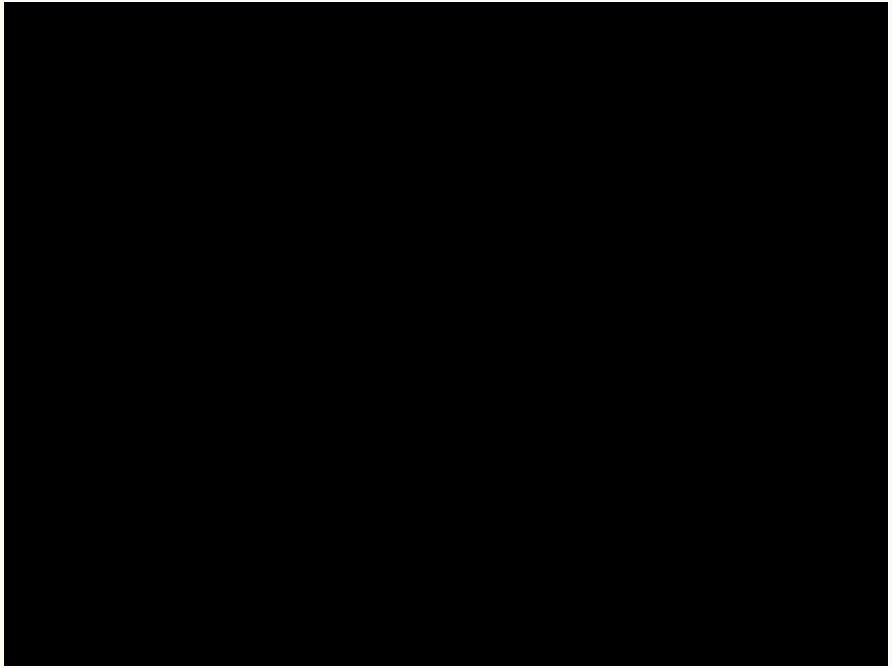


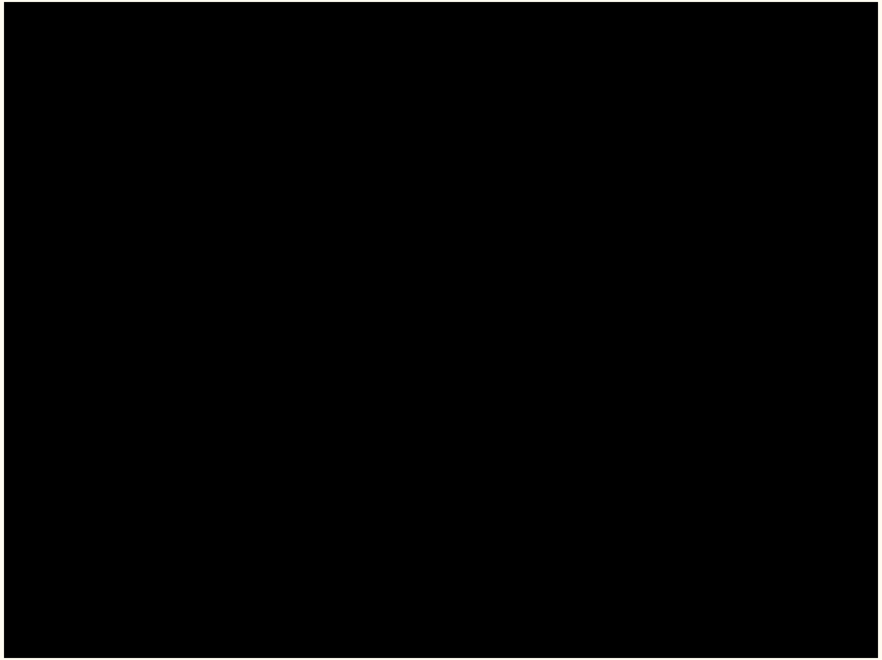
Physical Circuit Results



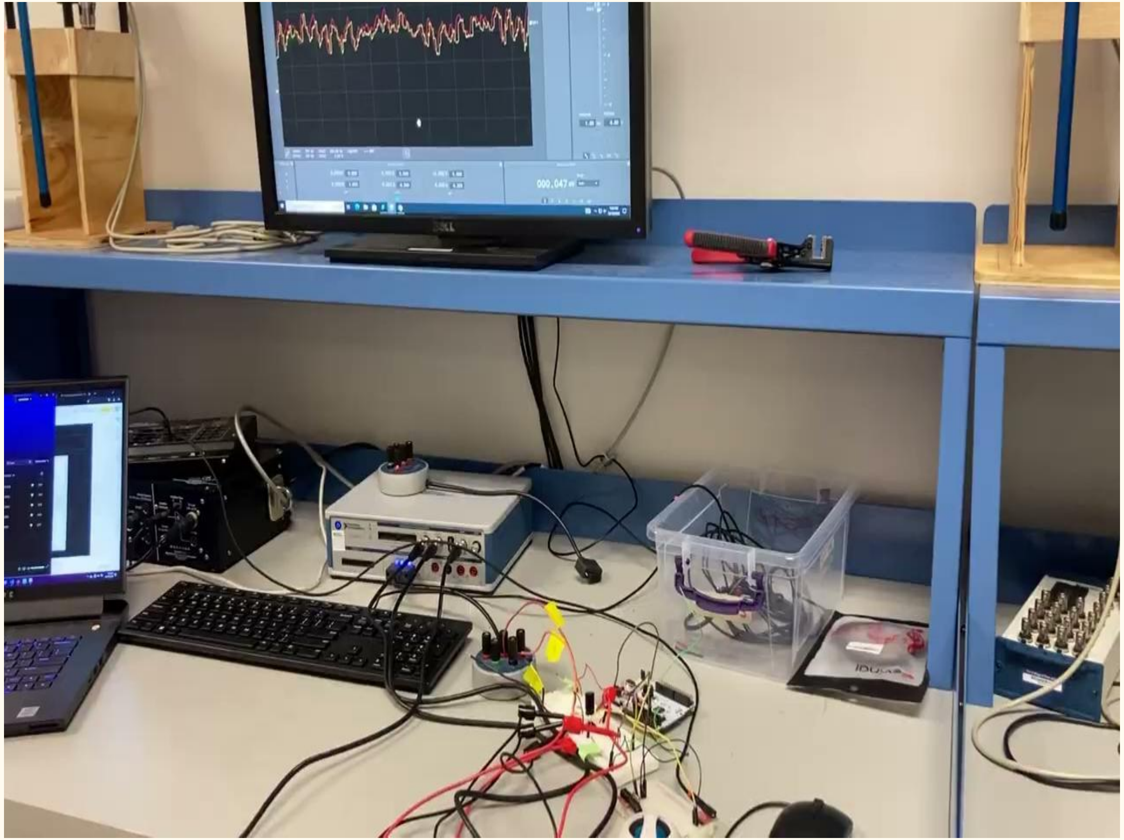


Notch Filter Video





Audio Demo



Conclusion

- Built and test signal conditioning circuit.
- Got experience using DSP, DAC, ADC, etc.
- Good coding experience.

Questions?